

“

*Aladdin is the #1 vendor in the Software Licensing
Authentication Tokens market for 2002 and 2003.*

”

— IDC Bulletin #31432, 2004 —

e A l a d d i n . c o m

HASP[®] SL

SECURE SOFTWARE DISTRIBUTION & LICENSING

HASP SL SDK Licensing Framework Guide

© Copyright 2005 Aladdin Knowledge Systems Ltd. All rights reserved worldwide. Aladdin, Aladdin Knowledge Systems, the Aladdin Knowledge Systems logo, HASP SL, HASP SL, HASP Commerce Server, HASP SL Storefront Engine, Purchase Builder, Usage Builder, HASP SL PortableStore Engine, Preview Systems, the Preview Systems logo, PortableStore, and Vbox are trademarks of Aladdin Knowledge Systems Ltd. All other trademarks are the property of their respective owners. Preview Systems' products and technologies are protected by U.S. patents and U.S. and international patents pending.

Revision 6.41.6

Table of Contents

The HASP SL Licensing Framework	1
Overview	1
How the HASP SL Licensing Framework Works	1
Runtime Component Relationships	3
HASP SL Licensing Framework Attributes	4
Components: Security and Origins	5
Components Defined	6
Execution Environment	6
Fallback UI	7
Install Packages	7
Keys	9
License	10
License Expression	11
License Generator	14
License Handler	14
License Handler UI	15
License State	16
Product	17
Product2LicenseFiles Binding	18
Protected File	19
ProtectedFile2License Binding	20
Workspace File	21
Using the Windows Reference Implementations	23
Introduction	23
What are the HASP SL Reference Implementations?	23
What do I do with the HASP SL Reference Implementations?	23
Getting Started	24
Installation	24
What is in the Reference Implementations?	25
Starting to Work with the SDK	26
Rename project files	27
Customizing a License Handler UI	28
About	28
To customize a License Handler UI	29
Localizing your client GUI files	32
Authoring DLLs for the HASP SL Framework	33
About	33
HASP SL Framework Compliance	34
License Handler Interface: ILICENSE2	35
Notes	35
Description	35
ILICENSE2 Interface Reference	36
Table of ILICENSE2 Calls	36
Initialize()	37
Start()	37
OkToRun()	38
InitialUserInteraction()	39
Charge()	40

Tick()	40
Receive()	41
Stop()	41
Finalize()	41
GetLastError()	42
Execution Environment: Interface and Services	43
IEE_SERVICES	43
Initialize()	43
Finalize()	43
GetServiceInterface()	43
GetLastError()	44
Execution Environment Services	44
Secure Storage Services Overview	45
SV_DISCRETE_STORAGE2	46
PF_Put	46
PF_Get	46
PF_Has	47
SV_CONTINUOUS_STORAGE2	48
SV_LOCAL_HIDDEN_STORAGE2	48
SV_LOCAL_EPHEMERAL_STORAGE	49
SV_TEST_STORAGE	50
Example Code:	50
SV_HARDWARE_FINGERPRINT	54
Example Implementation of Hardware Binding	54
GetHardwareFingerprint	54
GetNextFingerprint	55
SV_SOFTWARE_ACTIVATION	56
ActivatePerpetual	56
ComputeChallenge	58
CheckChallenge	58
GetOfferSecurity	59
GetOffers	59
SV_SOFTWARE_ACTIVATION_EX	60
GetPublishers	60
ActivatePerpetual	61
RecordTrialStart	61
ActivateTrial	62
GetServerVersion	62
ComputeChallenge	62
SV_SOFTWARE_ACTIVATION3	64
How do Aladdin's License Models implement activation?	64
ActivatePerpetual	65
ComputeChallenge	66
CheckChallenge	67
GetOfferSecurity	67
GetOffers	69
RecordTrialStart	69
ActivateTrial	70
GetServerVersion	70
RegisterUser	71
InitRegData	71
AddRegData	72
CommitRegistration	72
CustomerDataHas	73
CustomerDataGet	73

CustomerDataPut	74
SV_SOFTWARE_ACTIVATION3_EX	75
AddRegDataW	75
CheckPhoneChallenge	75
SV_DATE2	76
GetDate	76
GetBestDate	76
Add	77
Compare	77
Convert	78
SV_DESX	79
Process()	79
SV_DES	80
Process()	80
SV_HASH	81
Create()	81
Update()	81
Close()	81
SV_RANDOM	83
Generate()	83
SV_GUID	84
CreateGuid()	84
GetPrivilegeInstallationGuid()	84
SV_ERRORCONTEXTKEEPER	85
Open()	85
Set()	85
Get()	85
Close()	85
SV_MPI	87
Scan()	87
SV_SYSTEMLOCK	88
Open()	88
Lock()	88
Unlock()	88
Close()	89
SV_URS	90
LoadPublisherURSInterface()	90
UnloadPublisherURSInterface()	90
SV_EC_INFO2	92
GetECPath()	92
GetECCommandLine()	92
GetExpressionElement()	92
GetECInfo()	93
GetExpressionElementsForEC()	93
SV_LOGGER	94
Log	94

Example Interface between Handler and Handler UI95

Product Activation Sample UI Interface	95
prv_asm_lic_gui.dll	96
prv_asm_lic.dll	96
IASMLICENSEGUI	97
Initialize()	97
UIRun()	97
Finalize()	97
IASMLICENSEGUI_EX	98

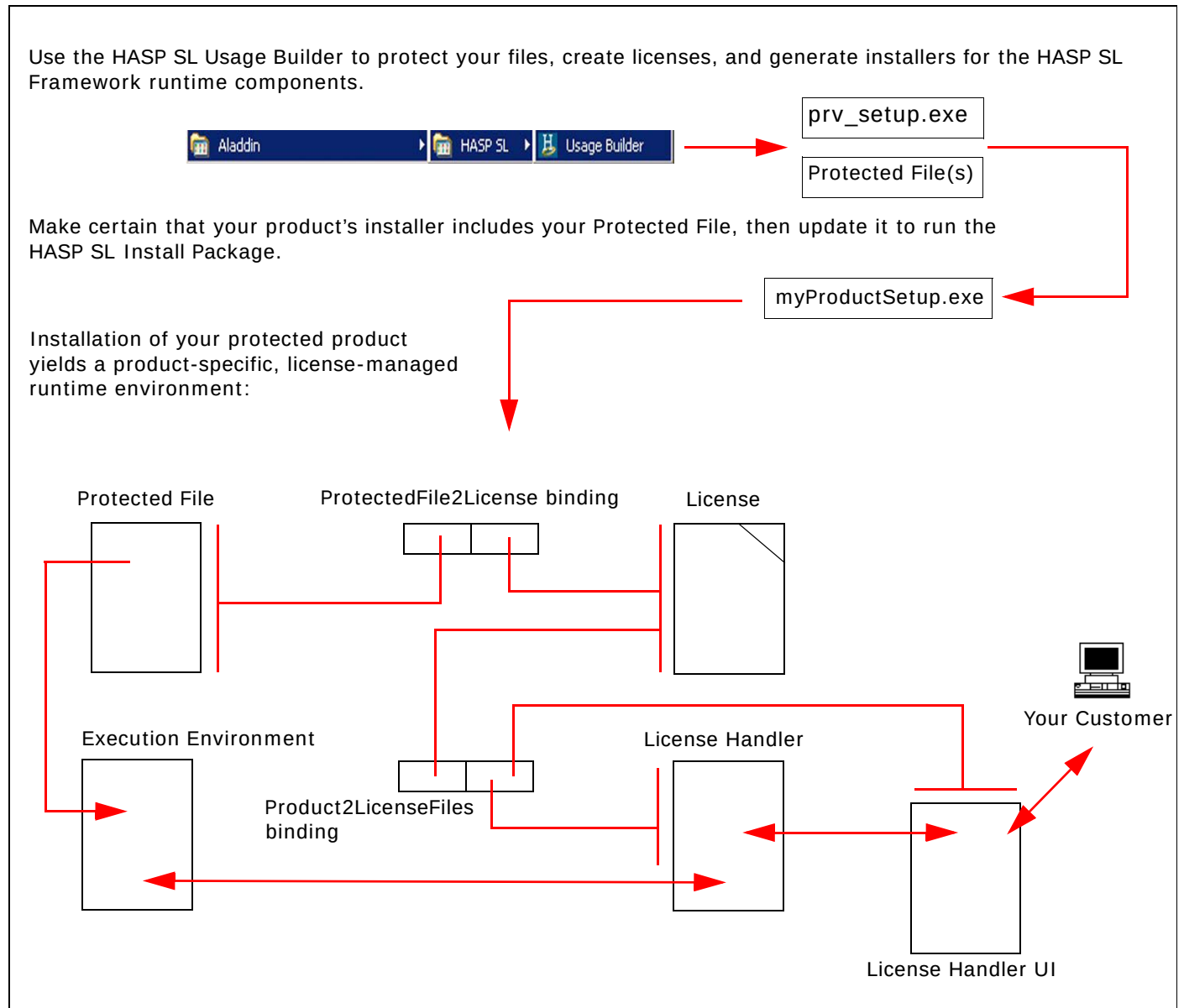
SetLicenseCallbackContext()	98
IASMLICENSEGUI2	99
SetCallbacks2()	99
IASMLICENSEGUI3	100
IASMLICENSEGUI_CALLBACKS	101
GetTrialState()	101
CheckPerpetualActivation()	104
CheckTrialActivation()	104
RegisterTrial()	105
GetPhoneChallengeCode()	105
CheckPhoneChallenge()	106
GetLastError()	106
IASMLICENSEGUI_CALLBACKS2	107
UserRegistered()	107
RegisterUser()	107
AddRegDataW	107
CommitRegistration	108
IASMLICENSEGUI_CALLBACKS3	109
UserRegisteredLocally	109
CheckPhoneChallengeWithLicData	109
Feature-based Licensing	110
Example Use Case: Eval and Full	110
Description of Licensing Requirements	110
How to Set Up your Product with an Eval and Full License	112
Runtime Description of the Eval and Full License	115
SDK Interface Loader	117
List of SDK Interface Loader Functions	117
ISDK Interface Reference	118
Overview	118
How does it work?	118
List of ISDK Functions	120
ISDK functions	121
Initialize()	121
Finalize()	121
GetContext()	121
GetEffectiveSet()	121
TransmitToLicenseDLL()	122
GetECInfo()	122
GetExpressionForEC()	123
GetLastError	126
License Data Implementation	127
Implementing License Data (Feature String)	127
Example	127

The HASP SL Licensing Framework

Overview

How the HASP SL Licensing Framework Works

Figure 1. HASP SL Framework Component Relationship Diagram



1. The HASP SL Framework is an enterprise solution for license-managed software.

- You create your own license terms, and the HASP SL Execution Environment enforces your license terms.
- License model reference implementations are provided for you to learn, as well as to customize and use. See [Using the Windows Reference Implementations](#).

2. At runtime, the HASP SL Framework is a product-specific, component-based, runtime environment, where components may be updated at anytime. The framework is comprised of Aladdin components (Execution Environment) and Publisher-defined components (injected product files and the governing license files associated with your product).
3. The license-managed runtime environment for a product is comprised of three types of components:
 - a. Protected Files

Portable Executable format (*.exe, *.dll, *.ocx) files from your product that are injected with Aladdin's security mechanisms.
 - b. Publisher-defined license components and Late Bindings

License components include the License, License Handler, License Handler UI, and any other publisher-defined functionality.

Licenses define terms of use governing your products. Bindings inform the Execution Environment of:

 - The License(s) that may govern a Protected File, and
 - The License Handler and License Handler UI that are to be used with the Product.

The license components and bindings are installed by the HASP SL setup file (<product>_prv_setup.exe), which is built in the Usage Builder.

Late Bindings occur as late as possible. Specifically, evaluation of the Licenses and Bindings occurs at the time that a Protected File is executed. Thus when you update your Licenses, you must also update the associated bindings.

The terms of use in a License are transparent to the Execution Environment, and are not constrained in any way by the HASP SL Framework.

In Aladdin's License Models, licenses that require activation map directly to an Offer created on the Activation Server Database. An Offer is used to define a batch of Activation Codes.
 - c. HASP SL Execution Environment

The HASP SL Execution Environment is also installed by the HASP SL setup file.
4. The Execution Environment supports any license-managed runtime environment. The Execution Environment:
 - a. Interprets Binding files and verifies components
 - b. Provides Services to the publisher-defined licensing components at runtime, such as Secure Storage, Hardware Fingerprinting, and Activation.
 - c. Is owned by Aladdin, and is backwards compatible for any custom license components that you implement.
5. License Handlers act as the directors to enforce licenses within a product-specific runtime environment. License Handlers:
 - a. Interpret the terms of use defined in a License.
 - b. Utilize the services provided by the Execution Environment, as needed, to enforce or otherwise handle the terms defined in a License.
 - c. Run the associated License Handler UI to display License status and solicit user input.

Three different Reference Implementations of a License Handler are provided. You may customize and use these for your own licensing purposes.

- 6.** License Handler UIs directly support the needs of a specified License Handler for a specific product.

The Reference Implementations include three different License Handler GUIs, each supporting the associated License Handler. This guide provides step-by-step instructions for updating the branding images and text for each of your products.

- 7.** The HASP SL Usage Builder is used to build your product-specific runtime environment as follows:
 - a.** Create and edit Workspace files, where your product definition information is created and stored.
 - b.** Inject Protected Files.
 - c.** Generate Bindings to associate your licensing components to each other, and to your products.
 - d.** Generate HASP SL setup files for your product-specific licensing components, associated bindings, and HASP SL Execution Environment.

Note that injected files are not included in the setup file, but are installed by your product installer with the rest of the product files.

Publishers may define their own License Models to be managed/enforced within the HASP SL Framework by creating their own licensing components that support the HASP SL API, then adding them to their product definitions via the HASP SL Usage Builder.

Runtime Component Relationships

- 1.** Product to Protected File
 - One Product may have many Protected Files.
- 2.** Protected File to License
 - One Protected File may be governed by many Licenses.
 - At runtime, Licenses are evaluated in the order you specify at build time in the HASP SL Usage Builder. The exception to this rule is that licenses (unconditional licenses) that do not require user intervention are evaluated first.
- 3.** License to License Handler
 - One License runs with one License Handler.
- 4.** License Handler to License State
 - Many Products using one License can have many License States.

HASP SL Licensing Framework Attributes

Attributes of the HASP SL Licensing Framework include the following:

1. Secure

The HASP SL Licensing Framework is tamper-proof and provides authentication security. See [Table 1 HASP SL Licensing Framework Components Data Table](#) for descriptions of individual components.

2. Policy-free as possible

License Handlers must support the HASP SL Licensing Framework interface, and a valid License must be installed in order to run a Protected File. Beyond that, anything may be extended or modified. Licenses and License Handler UIs are transparent to the HASP SL Licensing Framework; they are defined by publishers to fulfill specific licensing models.

3. Updateable runtime system

Components may be updated in any order, at any time. This means that you can change your licensing model without re-injecting *after* your product is installed. You can also update an injected file without modifying any licenses or license state.

4. 100% customizable and extendable

Reference Implementations of three HASP SL Licensing Models are provided in the SDK. If you don't like the way a particular aspect of the model is implemented, then you can change. The HASP SL Usage Builder provides the Runtime Libraries folder in which you can add your customized license components.

Components: Security and Origins

The table below lists the origins of each of the Licensing Framework components.

Table 1. HASP SL Licensing Framework Components Data Table

Component	Security	Publisher Replaceable ?	Description
Protected File	Injected	File: Yes Injector: No	This is the file (*.exe, *.dll, *.ocx) from your product that is injected by the HASP SL Usage Builder with Aladdin's security code. Note: HASP SL does not support the overlay sections of an overlay file.
License	Signed by Publisher's Workspace key	Yes	Created using a License Generator in the HASP SL Usage Builder. Licenses are text files containing the publisher-defined terms of use and the Offer/Server data required for Activation.
License Handler	Signed by Publisher's Workspace key	Yes	Created from the code provided in the Visual C++ reference implementation. The License Handler is a .dll (named by publisher) containing code to interpret a license, communicate with Execution Environment, and interface with License Handler UI. Publisher-defined License Handlers must support the HASP SL License Handler interface.
License Handler UI	Signed by Publisher's Workspace key	Yes	Created from the code provided in the Visual C++ reference implementation. The License Handler UI DLL (named by publisher) enables end-users to enter the information needed by the associated License Handler.
Fallback UI	Signed by Publisher's Workspace key	Yes	Created from the code provided in the Visual C++ reference implementation. The Fallback UI DLL (named by publisher) displays to end user when the Execution Environment fails to find a valid License Handler UI, or when the runtime environment has been tampered with. Publisher-defined Fallback UIs must support the HASP SL Fallback UI interface.
ProtectedFile2License Binding	Signed by Publisher's Workspace key	Yes	Binding file specified by publisher and generated by the HASP SL Usage Builder. This file binds the Protected File to the data in the License Expression: [License-Product-Active/Passive].
Product2LicenseFiles Binding	Signed by Publisher's Workspace key	Yes	Binding file specified by publisher and generated by the HASP SL Usage Builder. This file binds the Product-License pair to a License Handler and License Handler UI.
Execution Environment	Signed by Aladdin	No	Generated by the HASP SL Usage Builder. The Execution Environment evaluates bindings at runtime and verifies components. It also provides services to the License Handler, such as Storage (for example, Secure Storage), Fingerprint, Date/Time, and Activation.

Components Defined

Execution Environment

How the Execution Environment Works

The Execution Environment is included with the HASP SL Usage Builder, and built into the HASP SL [Install Packages](#) (<product name>_prv_setup.exe). It may be helpful to view [License Expressions: Runtime \(Figure 3\)](#) while reading the following description of the runtime Execution Environment.

At runtime, the OS loads the [Protected File](#), and the Protected File verifies the Execution Environment, and then loads it. The Execution Environment then retrieves the name of the publisher's [Fallback UI](#) to be used for this session.

The Execution Environment locates the appropriate [ProtectedFile2License Binding](#), then evaluates the [License Expression](#) inside the ProtectedFile2License binding to locate the effective set of local Licenses governing the Protected File.

For each element within the License Expression, the Execution Environment locates the appropriate [Product2LicenseFiles Binding](#). The Product2LicenseFiles binding contains the name of the [License Handler](#) and [License Handler UI](#) to be used with the Product-License combination, which the Execution Environment then validates.

How does the Execution Environment identify the effective License set?

In the scenario where a Protected File's License Expression contains more than one element, the HASP SL Execution Environment must decide which one(s) to use to allow the Protected File to run. This License (or combination of Licenses) is referred to as the effective License.

When a Protected File is asked to run, the Execution Environment performs the following operations to locate the effective license set.

For each process described, the Execution Environment evaluates each element within a License Expression in the order that it is defined at build time.

1. It evaluates each element in the License Expression to determine if the associated runtime components are installed and valid:
 - The License must be present and untampered (hash verified)
 - ProtectedFile2License binding for the Protected File must exist and the signature verified
 - License Handler signature is verified, and [Initialize\(\)](#) must indicate that it succeed
 - License Handler [Start\(\)](#) method must return true

If any of the elements fail this evaluation, then they do not make it into the effective set.

2. For each license in the remaining effective set, ask if the License Handler(s) will allow the Protected File to run unconditionally. If yes, then the License Handler will return TRUE in the "unconditional" parameter of [OkToRun\(\)](#). Unconditional usually indicates that one of the product licenses governing this Protected File has been activated. See [Table 1 HASP SL Licensing Framework Components Data Table](#).

If the License Handler(s) allows a Protected File to run unconditionally, then stop the evaluation because the effective license set is satisfied.

3. Evaluate each license in the effective set conditionally, by sending License Handler(s) another [OKToRun\(\)](#). For example, a condition may be that the end user must click OK to continue with the Trial.

Services Provided to the Runtime Environment

The Execution Environment provides the following services to the License Handler. See [Execution Environment: Interface and Services](#) for information on how to use these services to extend your License Handlers.

- Discrete Storage service
- Continuous Storage service
- Locally hidden Storage service
- Local Ephemeral Storage service
- Hardware Fingerprinting service
- Software Activation service
- Date and Time service
- Encryption/Decryption service
- Randomizing service
- Recursive, system wide named mutual exclusion lock service
- Load publisher interface service

Fallback UI

Summary

Produced by	Publisher. See reference implementation in SDK.
Build time location of product license instances	Runtime folder
Runtime installation by	prv_setup.exe. See Install Packages
Runtime loaded by	Execution Environment
Runtime location	HASP SL root folder
Runtime file name	Author defined

The Fallback UI is used exclusively by the Execution Environment to report errors to end-users when the normal License Handler UI is incapable of being used to report errors. The code for Aladdin's Fallback UI, prv_fallback.dll, is provided in the [Using the Windows Reference Implementations](#). It is expected that a publisher will have a single Fallback UI for all of their products.

Fallback UIs are defined for a Product at build time, and they are distributed in the [Install Packages](#). The Execution Environment knows which Fallback UI to use for the product-specific runtime environment because the name of it is included in the Protected File.

Install Packages

You generate Install Packages in the HASP SL Usage Builder after you have defined the necessary components of your Product. Install Packages are named <product name>_prv_setup.exe by default, but you may specify your own name and output location.

Install Packages never contain a [Protected File](#). The contents of install package depends entirely on how your product is defined, but usually contain your publisher-defined license components (License, License Handler, and License Handler GUI), the associated license binding, and the HASP SL Execution Environment. See the *HASP SL Usage Builder User Guide* for more information.

The HASP SL Framework is an updateable runtime system. The runtime system may only be updated with the HASP SL Install Packages. The following section lists what must be included in the install packages when updating specific components.

Updating Runtime Components

The table below describes what must be included in the install packages when updating runtime components.

Component to Update	Type of Update	Components to Include	Notes
Protected File	Replace	ProtectedFile2License Binding	Ship new Protected File and an Install Package. Protected Files are not included in the Install Packages.
License	Add	License ProtectedFile2License Binding Product2LicenseFiles Binding	When shipping new Licenses, always ship both bindings.
	Replace	License Product2LicenseFiles Binding	You may replace a License as long as the name of the file is the same.
	Remove	ProtectedFile2License Binding	Remove the License from the License Expression
License Handler	Replace	License Handler	As long as the filename does not change and the version is the same or higher, then all existing Product2LicenseFiles Binding automatically bind to the new License Handler.
	Add	License Handler	There is only one License Handler per Product-License, thus you would only add a new License Handler to handle a new License or Product.
	Remove	Product2LicenseFiles Binding	Assign a new License Handler to the Product-License, and the old handler is no longer used.
License Handler UI	Replace	License Handler UI	As long as the filename does not change and the version is the same or higher, then all existing Product2LicenseFiles Binding automatically bind to the new License Handler.
	Add	License Handler UI	There is only one License Handler UI per Product-License, thus you'd only add a new License Handler UI to support a new License or Product.
	Remove	Product2LicenseFiles Binding	Assign a new License Handler to the Product-License, and the old handler is no longer used.

Keys

Security keys are used by the HASP SL Licensing Framework to prevent runtime hacking. The following keys are visible to publishers.

Notation	Name	Type	Description
K_W	Workspace key	RSA variable length, Publisher-defined (512-2048 bits). Asymmetric	Generated by HASP SL Usage Builder when the workspace file is created. K_W is used to sign License files, Binding files, License Handler files, License Handler UI files, and Fallback UI files. K_W resides in your Workspace File , thus care should be taken to protect your workspace file. K_W maintains the integrity of the publisher-defined runtime environment. At runtime, objects signed with one K_W cannot be used by objects signed with a different K_W . For this reason, it is intended that publishers use a single workspace file to “HASP SL” their products.
K_L	License key	16 byte DESX Symmetric	Generated by HASP SL Usage Builder when a License is created, and placed in the License Expression and picked up by the Execution Environment at runtime. Publishers, however, may use K_L as they see fit. The HASP SL Framework provides K_L and the Execution Environment service SV_DESX to enable publishers to encrypt data on the local machine. This may be useful to publishers whose Licensing Model calls for data procured at runtime to be hidden from the user. Aladdin’s Product Activation License Handler reference implementation uses this key to encrypt the Activation Code.

License

Summary

Produced by	License Generator DLL
Build time location of product license instances	<product_dir>/Licenses
Runtime installation by	prv_setup.exe See Install Packages
Runtime loaded by	Execution Environment
Runtime read by	License Handler
Runtime location	HASP SL root folder
Runtime file name	Defined by HASP SL Usage Builder at build time

Description

The License is a publisher-defined entity. The HASP SL Licensing Framework does not know what a License is. To the [Execution Environment](#), the License is a transparent object. The Execution Environment only knows where it is installed and how to pass it on to the associated License Handler. The HASP SL Licensing Framework is designed to be as open, and policy-free as possible. Publishers may define the License, License Handler, and License Handler UI to be and do anything. The format of a License file does not matter as long as your License Handler can interpret it.

A license is a specific instance of a licensing model, and, after creation, cannot be modified. Licences are created in [Workspace Files](#) using HASP SL Usage Builder. You cannot access licenses in other workspaces. If you want to use a licence that was created in another workspace, then you must first export the license from that workspace and then import it into your workspace using HASP SL Usage Builder.

Licenses are installed on the end-user system via prv_setup.exe (not by a publisher's installer). See [Install Packages](#).

License Example

Licenses define the terms of use for a Product. For example, a License might be created for a trial period of 30-days maximum or 100 executions, whichever comes first. At the end of the trial period the software must be activated to continue to function. Once activated the software is usable without restriction for a period of one year.

Once such a license is defined it may be used by many different Products. The actual contents of the file that defines the License is known only to the software that produces the License at build time (License Generator) and the software that interprets the license on the end-user's machine (License Handler).

Aladdin License Generators

In Product Activation License Models, a License contains specific Offer data that is retrieved from the HASP SL Administration Server at the time of license generation. Each Offer is associated with a set of Activation Codes. See the *HASP SL Administration Server User Guide* for a description of how Offers and Activation Codes are created.

Aladdin's generators do not encrypt licenses. However, the HASP SL Licensing Framework does provide a [License key](#) and the [SV_DESX](#) runtime encryption service for publishers who wish to encrypt Licenses or any other data.

License Expression

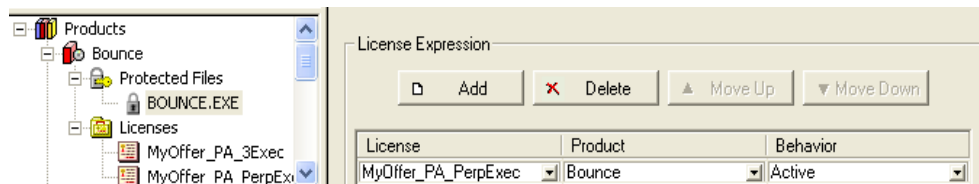
The [License Expression](#) is created using the HASP SL Usage Builder and contained in the [ProtectedFile2License Binding](#), which is distributed and installed by HASP SL [Install Packages](#)

A Protected File, even if shared by more than one product, has only one License Expression at runtime. The License Expression defines the set Licenses that may govern a specific Protected File.

Build Time

In the HASP SL Usage Builder, you create License Expressions for each Protected File ([License Expressions: Build Time Example, One Element \(Figure 2\)](#)) by creating a set of rows, or elements, where each element defines a possible [License + Product + Behavior] that may govern your Protected File at runtime.

Figure 2. License Expressions: Build Time Example, One Element



Runtime Description

At runtime, the purpose of the Execution Environment is to identify a single license which permits the Protected File to run. If the Execution Environment fails, then the Protected File may not run.

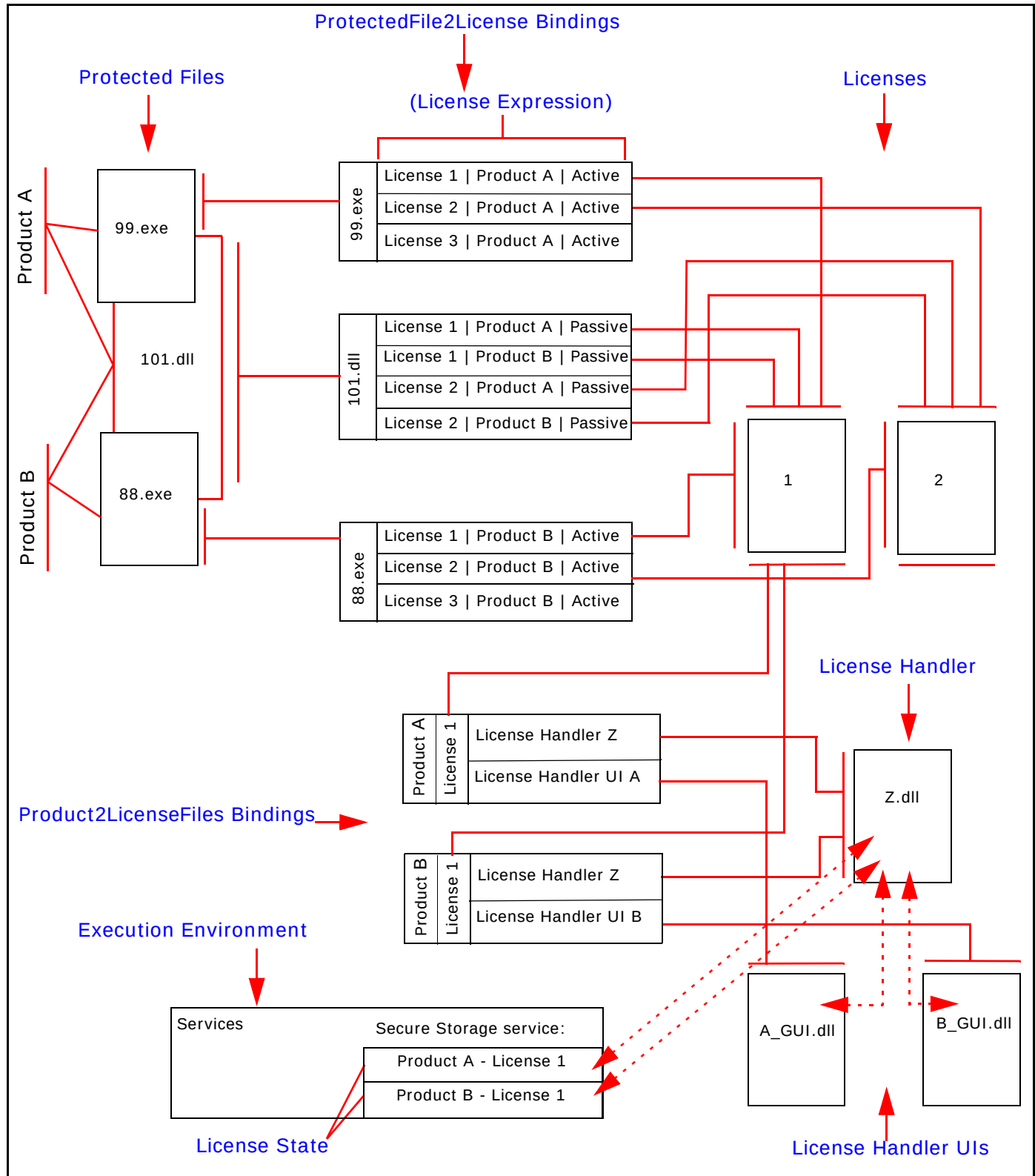
While there may be more than one valid License in the runtime environment, only one license is used to permit the Protected File to run. This license is referred to as the effective license. See [How does the Execution Environment identify the effective License set?](#)

The [License + Product] parameters in License Expression define the [License State](#) location where Aladdin's License Models check and subsequently charge usage for actively consumed Licenses.

Note also that Aladdin's License Models use the Behavior parameter in the License Expression to determine whether to charge (Active) usage of the specified License to the License State or not (Passive). You may modify these behaviors in your License Handler, or create your own behaviors in addition to these.

[License Expressions: Runtime \(Figure 3\)](#) is an example of how the License Expression fits into the runtime system. A description of the diagram follows.

Figure 3. License Expressions: Runtime



Runtime Example

The example [License Expressions: Runtime \(Figure 3\)](#), shows two products: Product A, and Product B. These products each have their main executable (99.exe and 88.exe respectively) protected, and additionally they share a single Protected File called 101.dll. There is one ProtectedFile2License binding for each Protected File, which contains the License Expression defining the set of Licenses that govern the Protected File.

Interpreting Product A's License Expression

Three Licenses govern Product A: License 1, License 2, and License 3. License 3 is defined in the License Expression, but, as you can see in [License Expressions: Runtime \(Figure 3\)](#), it is not installed. Thus, the Execution Environment considers License 3 as invalid.

License 2 is installed, but the Product2LicenseFiles binding for License 2 is not installed, and thus the Execution Environment would determine License 2 to be invalid as well.

License 1 is the only valid License in this example. In the case where there is more than one valid License, the Execution Environment must identify one of these and use it to permit the Protected File to run. See [How does the Execution Environment identify the effective License set?](#)

When 99.exe is run by the end user, the Execution Environment evaluates the License Expression and determine that License 1 is the only valid License. The Execution Environment then evaluates the Product2LicenseFiles binding for [License 1-Product A] to find the License Handler for License 1. The Execution Environment then passes the data from the two bindings to the License Handler. This data is all the information the License Handler needs to do it's job.

- [License 1 - Product A] defines the License State where usage of 99.exe is charged.
- [Active] indicates that usage of terms in License 1 are to be actively charged. Or, put another way, the terms in License 1 are consumable.
- License Handler GUI A is displayed to handle end user interaction for [License 1 - Product A].

Shared DLLs Should Always be Passive

Let's look at the shared DLL 101.dll. In order for 101.dll to be used by Product A and Product B, the License Expression for 101.dll must contain elements for each Product for each License as defined in [License Expressions: Runtime \(Figure 3\)](#). In this example, 101.dll always runs and is never charged for consumed usage because the [Product - License] combination is set to Passive.

What would happen if the License Expression for 101.dll defined License 2 for 88.exe to be Active? When Product B runs both 88.exe and 101.dll, the usage terms are consumed twice as fast because License State is charged per [Product - License], and **not** per [Protected File - License]. Thus the License Handler charges usage at the [Product - License] License State when 88.exe is run, and it charges usage at the [Product - License] License State when 101.dll is run.



Services should be set to Passive as the Microsoft Platform does not support a GUI for services. Services therefore do not consume executions when they are run. If you do want a Service to be set to Active, and therefore consume usage, you must use the SDK described in this document.

License Generator

Summary

Name of License Generator provided by Aladdin	Product Activation License generator: prv_asm_lic_gen.dll Subscription License generator: prv_subs_lic_gen.dll Try & Die License generator: prv_td_lic_gen.dll The source code is provided in the Reference Implementations.
Used by	HASP SL Usage Builder

Description

The License Generator is a build time component that generates licenses.

Each license type has a License Generator that is dynamically loaded by the builder when creating or editing a license instance.

The License Generator must be installed into the Runtime Library folder in the HASP SL Usage Builder in order for the HASP SL Usage Builder to use it.

License Handler

Summary

Produced By	Publisher or Aladdin
Build time location	Runtime folder
Runtime installation by	prv_setup.exe. See Install Packages
Runtime loaded by	Execution Environment
Runtime location	HASP SL root folder
Runtime file name	Author defined

Description

The runtime environment uses a License Handler to interpret the data in a license. Each Protected File loaded under the control of the HASP SL Framework results in the loading of one or more License Handlers, depending on the valid Licenses for that Protected File. There is a one to one relationship between License and License Handler.

A License Handler's primary responsibility is to determine if a Protected File should or should not be allowed to execute. This determination is performed by querying each of the Licenses governing the Protected File. In a [Feature-based Licensing](#) model, your License Handler responds to requests to run from inside the Protected File.

In general a License Handler is not expected to have any user interface because that role is delegated to a subsidiary called the License Handler UI. However, there are no rules, and your License Handler may also include the GUI necessary to support it.

License Handlers must be installed in the Runtime Library folder in the HASP SL Usage Builder, in order to be used by the HASP SL Usage Builder. License Handlers are installed on end user's machines via [Install Packages](#) (prv_setup.exe), and not by the product's installer.

License Handler UI

Summary

Produced By	Publisher or Aladdin
Build time location	Runtime folder
Runtime installation by	prv_setup.exe. See Install Packages
Runtime loaded by	License Handler
Runtime location	HASP SL root folder
Runtime file name	Author defined

Description

The License Handler UI is responsible for managing interaction with the user regarding all licensing interaction, including presenting nag windows, purchasing activation codes or accessing publisher customer support services. As it is the primary interface for the end-user, nearly all publisher or vendor-branding information is contained in this library.

Due to the fact that the License Handler UI provides services only to the License Handler, the actual interface between them is not a HASP SL Framework definition, but rather a relationship that is publisher-defined.

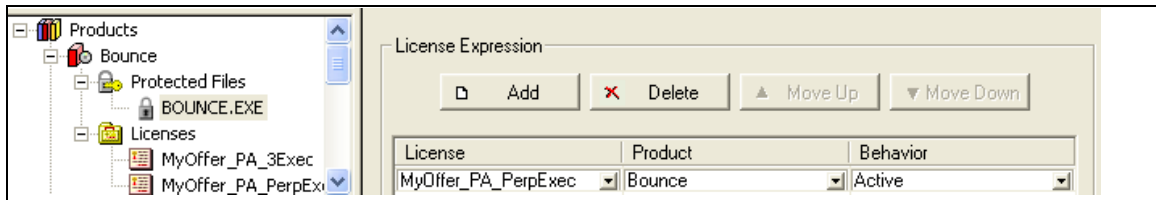
License Handler UIs are installed on end user's machines via [Install Packages](#) (prv_setup.exe), and not by the product's installer.

See also [Product Activation Sample UI Interfaces \(Figure 9\)](#).

License State

License state is stored at the address defined as [License-Product], where License-Product is derived from the [License Expression](#) of the Protected File. Publishers define the License State address for a Protected File when they create the License Expression ([License Expression Editor in the HASP SL Usage Builder \(Figure 4\)](#)).

Figure 4. License Expression Editor in the HASP SL Usage Builder



The [Execution Environment](#) passes the License State address [License-Product] to the License Handler so that it can check the usage state of active Licenses.

Aladdin's Reference Implementations store the License State of active trial Licenses using the Secure Storage services provided by the Execution Environment. See [Secure Storage Services Overview](#). The Reference Implementations compare the terms in a License with the License State for a given product to determine whether or not the Protected File may be run.

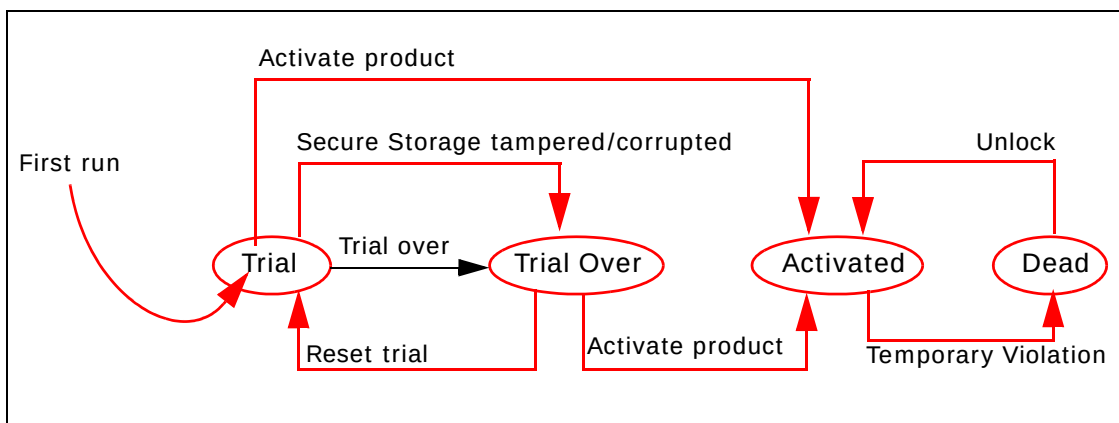
Possible License States at Runtime

The License Handler uses the Secure Storage provided by the Execution Environment to store the License State. See [Secure Storage Services Overview](#) for more information.

The Reference Implementations define the flow of the GUI based on the existing state plus user input. When a Protected File is run for the first time, the License Handlers set the initial state to Trial prior to allowing it to run.

The following state diagram and state table describe the license states Aladdin's Product Activation model.

Figure 5. License State Diagram



State	Input	Next State
None	Run Protected File for the first time	Trial
Trial	Activate (phone or internet)	Activated
	Trial ends	Trial Over
	Tamper with or HASP SL Framework, storage corrupted	Trial Over
Activated	Temporary violation	Dead
Trial Over	Trial Reset	Trial
	Activate (phone or internet)	Activated
Dead	Phone unlock	Activated

Product

Publishers define Product names in HASP SL Usage Builder during build time. The HASP SL Usage Builder generates an ID for the publisher-defined Product name, which is used by the HASP SL Licensing Framework to track [License State](#), and also to bind a License Handler UI to a License. See [Product2LicenseFiles Binding](#).

You may rename your Product in the Usage Builder, without breaking it's runtime compatibility.

Product2LicenseFiles Binding

Summary

Produced By	HASP SL Usage Builder
When is update required?	When you are updating a customer's runtime environment, you must generate a new binding whenever you add a new License. You will also need to update your binding if you rename your License Handler or License Handler UI. See Updating Runtime Components .
Build time location	<product_dir>/License
Runtime loaded by	Execution Environment
Runtime installation by	prv_setup.exe. See Install Packages
Runtime location	HASP SL root folder
Runtime file name	Defined by HASP SL Usage Builder at build time.

Description

The Product2LicenseFiles binding defines the License, License Handler, and License Handler UI to be used at runtime for a specific Product. This data is defined by the publisher in the HASP SL Usage Builder at build time, then generated by the HASP SL Usage Builder and included in the [Install Packages](#).

For each License, there is usually a single License Handler defined to handle it. However, for each product, there may be a product-specific License Handler UI defined.

Thus the Product2LicenseFiles binding binds the following data: [Product ID, License ID, License Handler ID, License Handler UI ID].

[License Expressions: Runtime \(Figure 3\)](#) illustrates two Product2LicenseFiles bindings: [Product A-License 1] and [Product B-License 1]. These bindings illustrate two Products sharing a single License and a single License Handler, yet using different License Handler UIs.

Protected File

Summary

Produced By	Publisher
Post-injection build time location	<product_dir>/CodeItem
Runtime installation by	Publisher (Product's installer)
Runtime loaded by	Operating System
Runtime location	Publisher defined (installed with Product)
Runtime file name	Publisher defined

A Protected File is a file that has been injected with Aladdin protection code. You choose the files from your product to be injected. In order to protect a file, it must be:

Windows: Portable Executable format (*.exe, *.dll, *.ocx)

Macintosh: Carbon PEF files

Injection updates the time stamp of a file to the time of injection.

Protected Files are the starting point for Aladdin protection. Protected Files are not injected with any License or GUI information. All associations are handled outside the Protected File via bindings, enabling you to update your licensing model, branding schema, or the HASP SL Execution Environment at any time without re-injecting your product files.

Installation of Protected Files is the responsibility of the product's installer. There are no restrictions on where a Protected File may be installed.

ProtectedFile2License Binding

Summary

Produced By	HASP SL Usage Builder
When is update required?	When you are updating a customer's runtime environment, you must generate a new binding whenever you add a new License, or replace your Protected File.
Build time Location	<product_dir>/EC
Runtime installation by	prv_setup.exe
Runtime read by	Execution Environment
Runtime location	HASP SL root folder
Runtime file name	Defined by HASP SL Usage Builder at build time.

Description

A ProtectedFile2License binding binds a Protected File to the set of Licenses that govern the Protected File. This set of Licenses is defined specifically in the License Expression, which resides inside the ProtectedFile2License binding. See [License Expression](#).

For each Protected File, there can only be one ProtectedFile2License binding, and consequently only one License Expression. You must generate and install a new ProtectedFile2License binding whenever:

- You generate and install a new Protected File.
- You generate and install a new License for the Protected File.

See also [Updating Runtime Components](#).

Workspace File

myWorkspace.prv

The workspace file contains the [Workspace key](#) and every care should be taken to prevent the file from leaving the publisher's control.

Summary info

Produced by	HASP SL Usage Builder only
File type	Text file
Used by	HASP SL Usage Builder
Key used	RSA variable key length (512-2048 bits). The Publisher defines the length of this Key in the Preferences dialog box from the Tools menu in the HASP SL Usage Builder. The HASP SL Usage Builder generates the Workspace key when a new workspace is created, and stores it in the Workspace file (.prv). This key is used to sign components at build time.

Description

The workspace file is a project file generated by the HASP SL Usage Builder. When the HASP SL Usage Builder creates a new workspace, it generates an RSA key of publisher-defined length (512-2048 bits). The Workspace file uses this key to sign the publisher-defined components defined for each product. If you do not want to use this key, you can import any RSA key using HASP SL Usage Builder.

Runtime environments defined for Products are bound by workspace file keys. All runtime descriptions are understood to be unique per key, thus preventing any conflicts between publishers or products generated in other workspaces.

The HASP SL Usage Builder is the Workspace file editor. The Workspace file is used to define the product-specific runtime environments to be managed by the HASP SL Framework.

See the *HASP SL Usage Builder User Guide* for procedures on creating Products in your workspace files.

Using the Windows Reference Implementations

Introduction

What are the HASP SL Reference Implementations?

The HASP SL Reference Implementations are the MS Visual C++ .NET source developer solutions for the HASP SL License Models provided with the Usage Builder:

- Product Activation License Model
- Try & Die License Model
- Subscription License Model

The HASP SL Reference Implementations are working examples of the [The HASP SL Licensing Framework](#). Examine them to understand the HASP SL Licensing Framework, and modify them as necessary to create your own licensing models.

What do I do with the HASP SL Reference Implementations?

Modify the reference implementations as you see fit, then use them as you would your own code. We've identified the following four levels of customization that publishers may implement to meet their licensing needs.

This guide provides examples of Level I customizations. It also provides the API documentation to assist you in implementing the remaining levels of customization.

The table below suggests the type of customizations that publishers may be interested in.

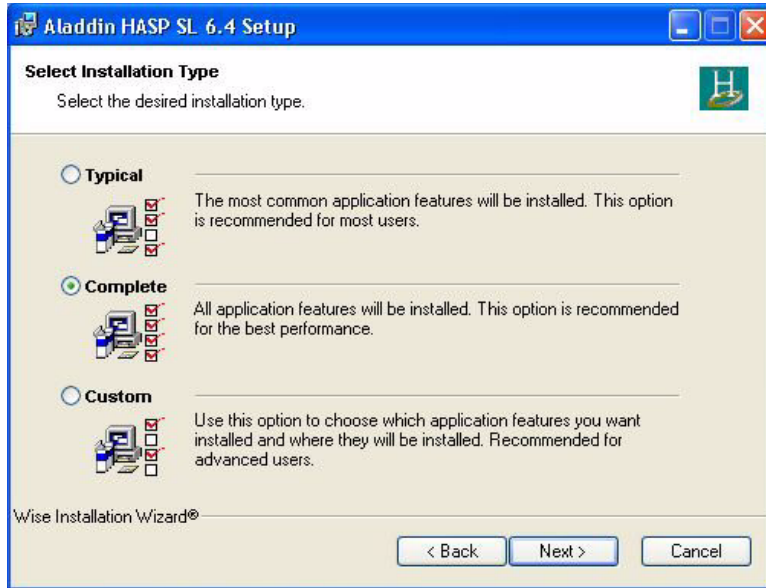
Table 2. Levels of Customization

Level	Type	Purpose	Notes
I	Required	Publisher branding/marketing	Considered cosmetic, the following step-by-step procedures are provided to make Level I customizations: To customize a License Handler UI
II	Optional	You like Product Activate Licensing Model, for example, but you want to tweak the flow to fit your needs.	Examples include: Change flow of GUI displayed to customer. See Product Activation Sample UI Interfaces . Remove "Trial" to make Activate only model.
III	Advanced	You want to keep the Product Licensing Model, for example, but you want to remove, modify, or extend a fair amount of functionality	Examples include: Modify state information stored. Store data in secure storage. Add user registration. Publisher wants to shut down a running Product. Metering schemes. Add fingerprint protection.I
IV	Expert	Create new License Models	New License Models may include rentals, subscriptions, calendar-restricted usage, etc.

Getting Started

Installation

To install the HASP SL Reference, choose the **Complete** option when installing the Usage Builder:



You can select the **Custom** option, and choose to install with or without the tutorials.

The HASP SL Reference Implementations will be installed in the following location:
<installation folder>\HASP SL\Client SDK.

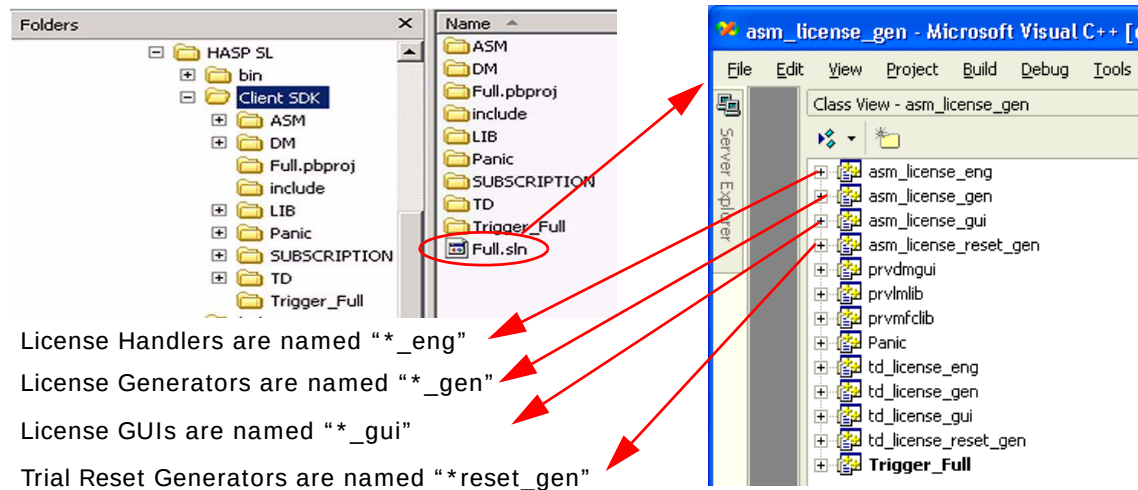
By default, <installation folder> = C:\Program Files\Aladdin. See [Using the Windows Reference Implementations](#) for a description of the installed project files.

What is in the Reference Implementations?

You may open all of the reference implementation projects by double-clicking the SDK solution file, which is in the following location by default:

C:\Program Files\Aladdin\HASP SL\Client SDK*.sln. See [Figure 6](#). Your *.sln file will be named full.sln or toolkit.sln, depending on which installation of HASP SL you have.

Project Filename (relative to ..\HASP SL\Client SDK\)	License Model	Project Description
*.sln	Product Activation	• asm_license_eng = License Handler • asm_license_gen = License Generator • asm_license_gui = License Handler UI • asm_license_reset_gen = Trial Reset Generator The Product Activation model provides most of the features, including: perpetual activation, trial activation, purchase URL, trial-start notification, and the define-your-own trial period. Use this Reference Implementation when activating against the Commerce Server, v. 6.3 or later.
	Try & Die	• td_license_eng = License Handler • td_license_gen = License Generator • td_license_gui = License Handler UI • td_license_reset_gen = Trial Reset Generator The Try & Die model provides only the define-your-own trial period feature, and has no server activation or notification.
	Subscription	• subs_license_eng = License Handler • subs_license_gen = License Generator • subs_license_gui = License Handler UI • subs_cancel_gen = License Cancellation Generator • subs_cancel_tool = License Cancellation Tool The Subscription model provides numerous feature, including: grace periods, server-side expiration dates with options for subscription renewal. A cancellation utility is also provided to ensure that usage of the software is prevented where the subscription has been cancelled.
	Aladdin Fallback UI	• Panic Use this Reference Implementation to create your custom publisher Fallback UI. See Fallback UI.

Figure 6. Example of project location

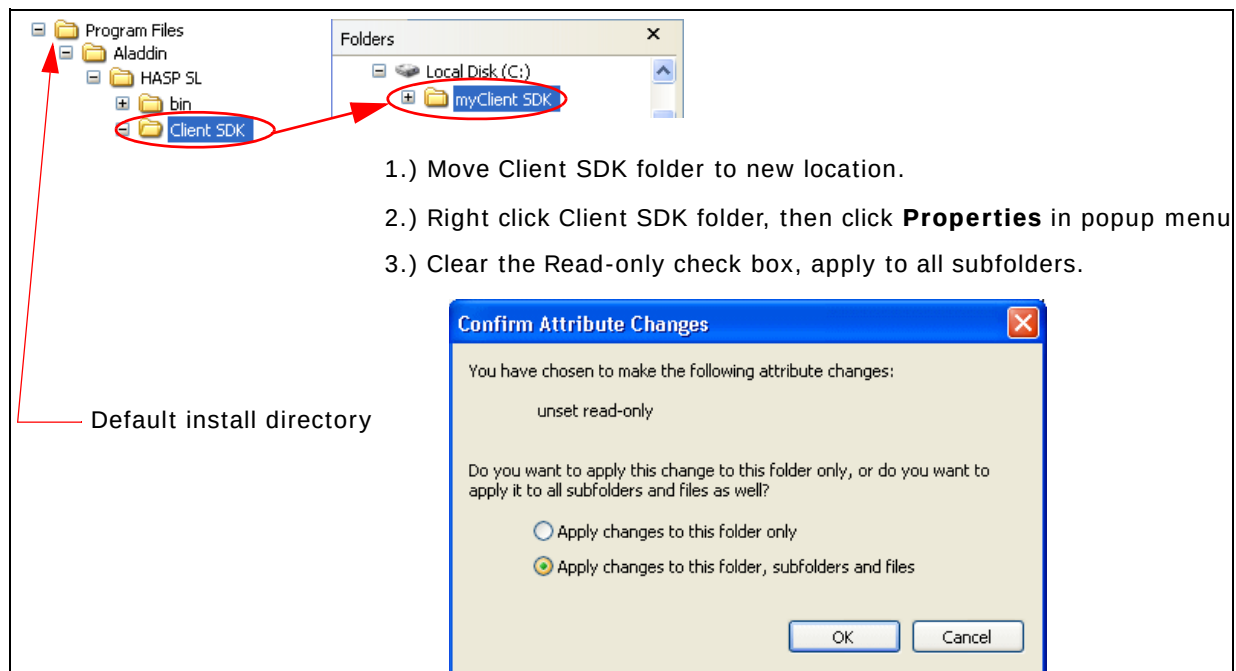
Starting to Work with the SDK

The Visual C++ .NET project files are set to read only. Prior to using the projects, copy the Client SDK folder to a new location and remove the Read-only permissions.

Default reference implementation location:

Windows: C:\Program Files\Aladdin\HASP SL\Client SDK*.sln

Macintosh: C:\Program Files\Aladdin\HASP SL\Client SDK\Full.pbproj\project.pbx.proj

Figure 7. Copy SDK folder to new location

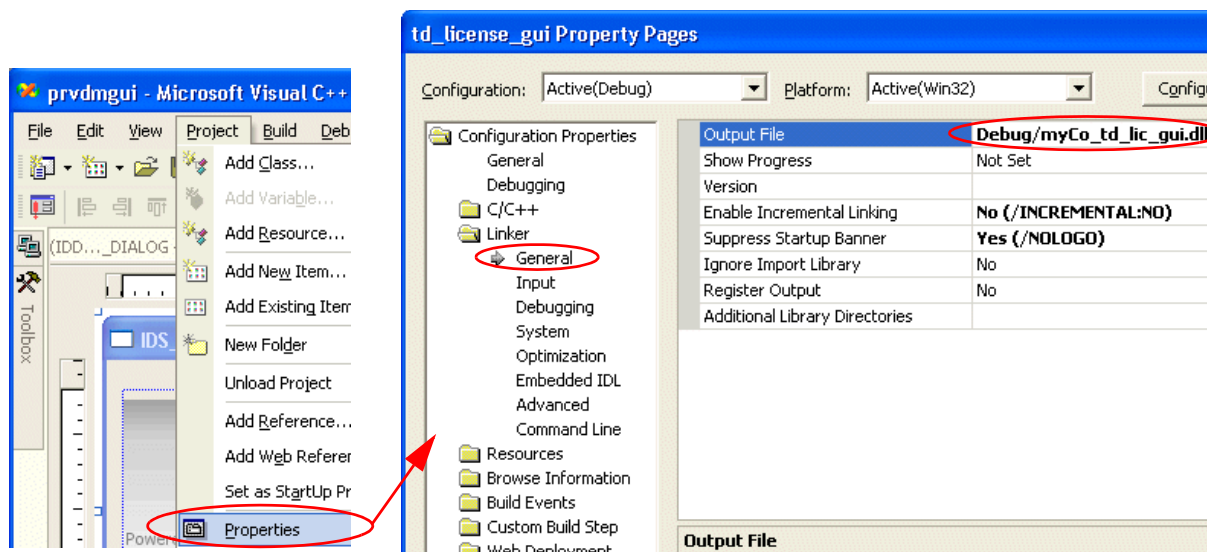
Rename project files

We recommend that you change the name of the Aladdin project files to make them your own. This will ensure that when you add your files to the Runtime Library in the Usage Builder, they will not overwrite the default Aladdin files. Follow the procedures below to rename your files.

To rename the output files

Repeat the steps below for each project. If you are updating your own custom DLL, then you at least need to increment the *Version* value.

1. In the Visual Studio .NET main menu, choose **Project > Properties**. The Project Properties dialog displays.



2. In the Project Properties dialog, select **Linker > General**, then enter a value in the *Output File* field at right.

For example, rename "prv_td_lic_gui.dll" to "myCO_td_lic_gui.dll."

3. The Version number is obtained from the version header files.



The version is controlled by version header files. The version header files are located with all the other source files at each project's directory. Their name is constructed from the project name + "version" (for example: asm_license_eng -> asm_license_engVersion.h).

In each header file simply change the version number twice (once as a number and once as a string):

```
#define __AKS_NUMERIC_VERSION 1,0,0,1 -> #define __AKS_NUMERIC_VERSION 5,5,7,2
#define __AKS_STRING_VERSION "1,0,0,1" -> #define __AKS_STRING_VERSION "5,5,7,2"
```

4. Click **OK** to close the dialog box and save your changes.

Customizing a License Handler UI

This procedure applies to customizing the Aladdin Fallback UI as well.

About

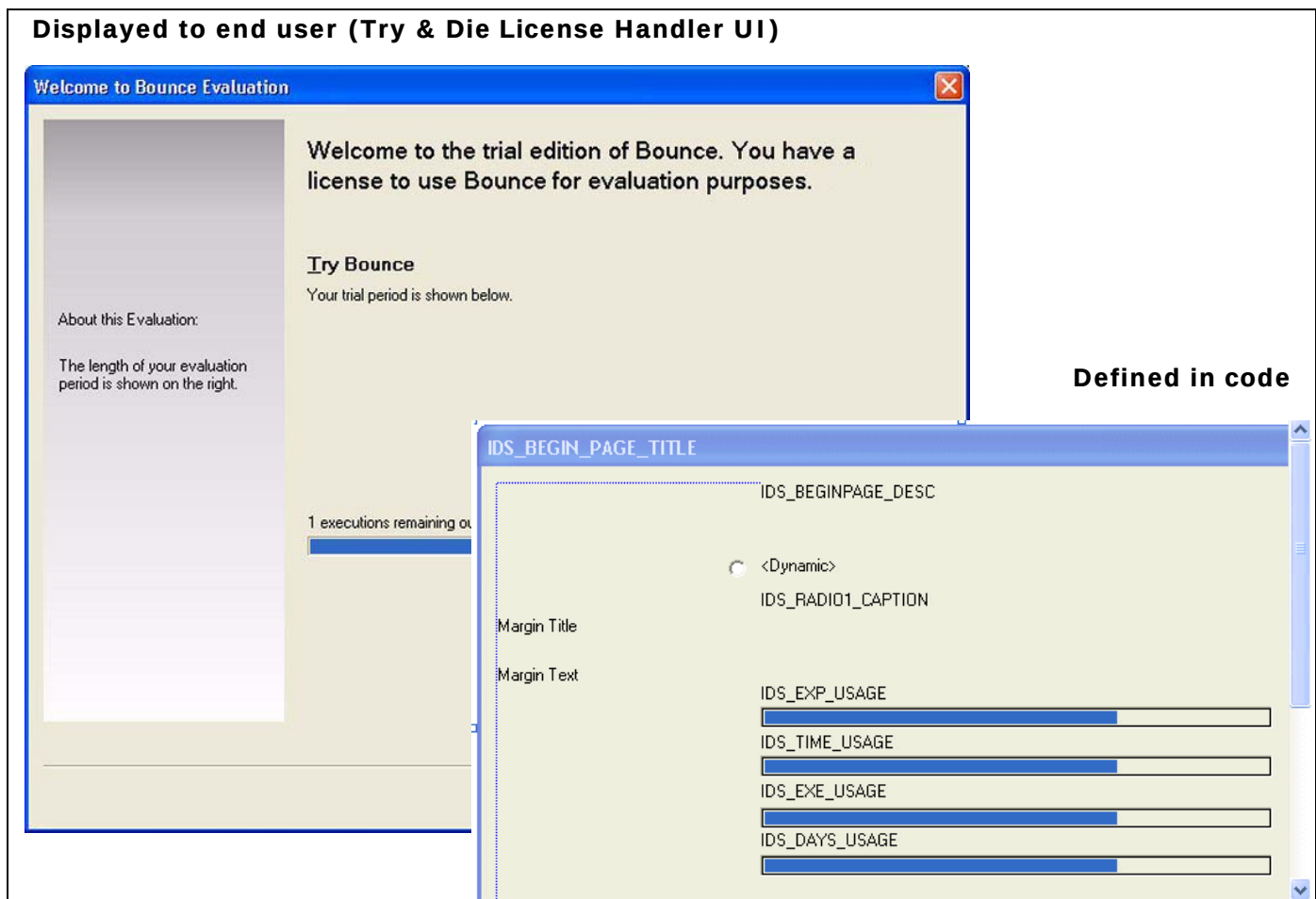
The License Handler UI defines the “user experience” in terms of the GUI that is displayed to your users. The License Handler UI supports the functionality defined in the associated License Handler. See [What is in the Reference Implementations?](#) to see where the License Handler GUI projects are located.

You may create product-specific License Handler UIs, incorporating product branding to be displayed to your end users, or you may choose to create a more generic License Handler UI to display for all of your products.

Remember, License Handler UIs are bound to unique Product-License combinations. See [License Expressions: Runtime](#). See also [License Handler UI](#).

The following figure shows you what the License Handler UI looks like in the Execution Environment on the end user’s desktop, and from the SDK developer’s point of view.

Figure 8. License Handler GUI example



To customize a License Handler UI

Open the License Handler UI project that you want to customize. See [What is in the Reference Implementations?](#). This procedure uses Aladdin's Try & Die License Handler UI as an example.

Step 1: Update the Version resource

The version is controlled by version header files. The version header files are located with all the other source files at each project's directory. Their name is constructed from the project name + "version" (for example: asm_license_eng -> asm_license_engVersion.h).

In each header file simply change the version number twice (once as a number and once as a string):

```
#define __AKS_NUMERIC_VERSION 1,0,0,1 -> #define __AKS_NUMERIC_VERSION 5,5,7,2
#define __AKS_STRING_VERSION "1,0,0,1" -> #define __AKS_STRING_VERSION "5,5,7,2"
```

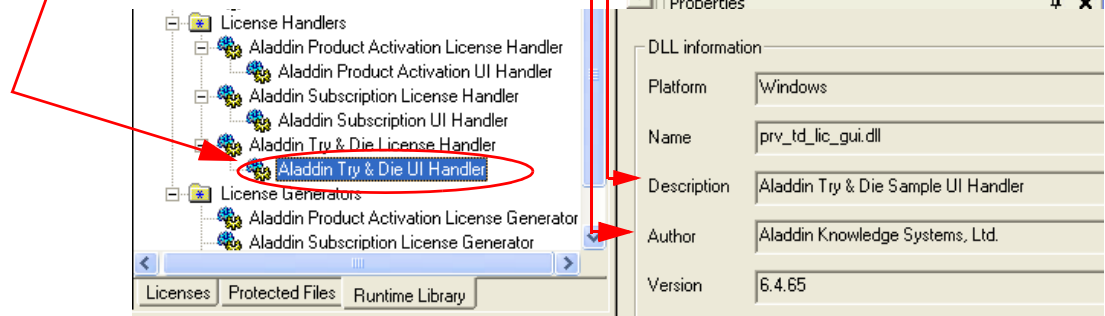
Step 2: Update the name of the Runtime Library

From the Source Files folder, open your project's *.cpp file (for example, td_license_gui.cpp). In the right pane, enter new values for the PRV_URS_IMPLEMENT_DLL macro:

```
RS map
URS_MAP_BEGIN()
PRV_URS_MAP_ENTRY(ITDLICENSEGUI_NAME, s_ITDLICENSEGUI)
V_URS_MAP_ENTRY(ITDLICENSEGUI_EX_NAME, s_ITDLICENSEGUI_EX)
URS_MAP_END()
```

```
RS implementation
```

```
URS_IMPLEMENT_DLL("Aladdin Try & Die UI Handler",
                  "Aladdin Try & Die Sample UI Handler",
                  "Aladdin Knowledge Systems, Ltd.")
```

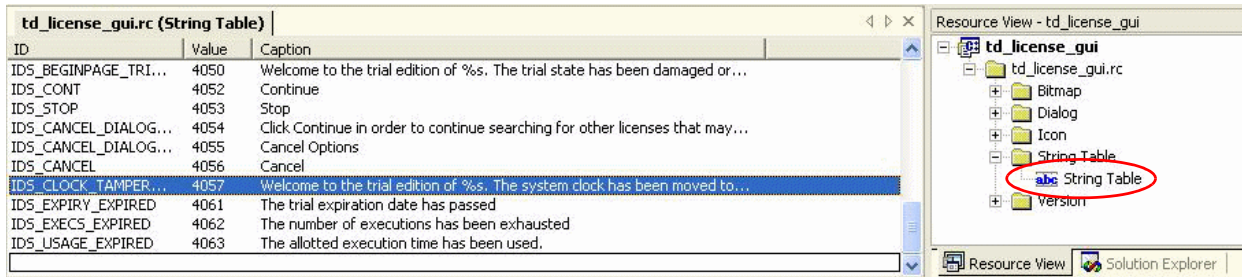


This macro specifies three values for your runtime component: display name, description, author. These values display in the Usage Builder when you add your DLLs to the Runtime Library tab of your workspace.

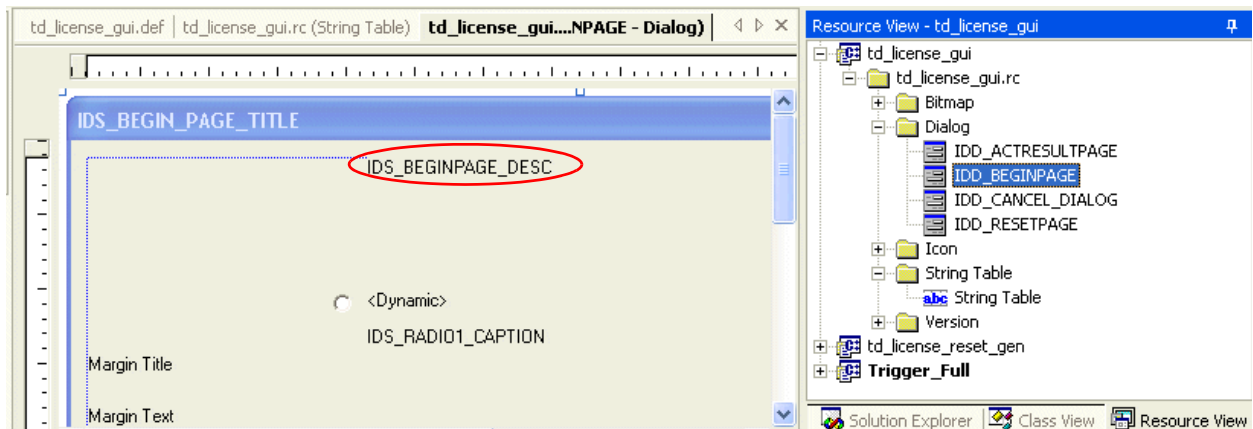
Step 3: Update the UI text displayed to end users

UI text is defined in a Text String table, making it easy to update or translate.

Open the String Table in your License Handler UI project, then modify the display text as necessary. For example:



To determine where and when the text strings display to your users, open the dialogs and view the text IDs. For example:



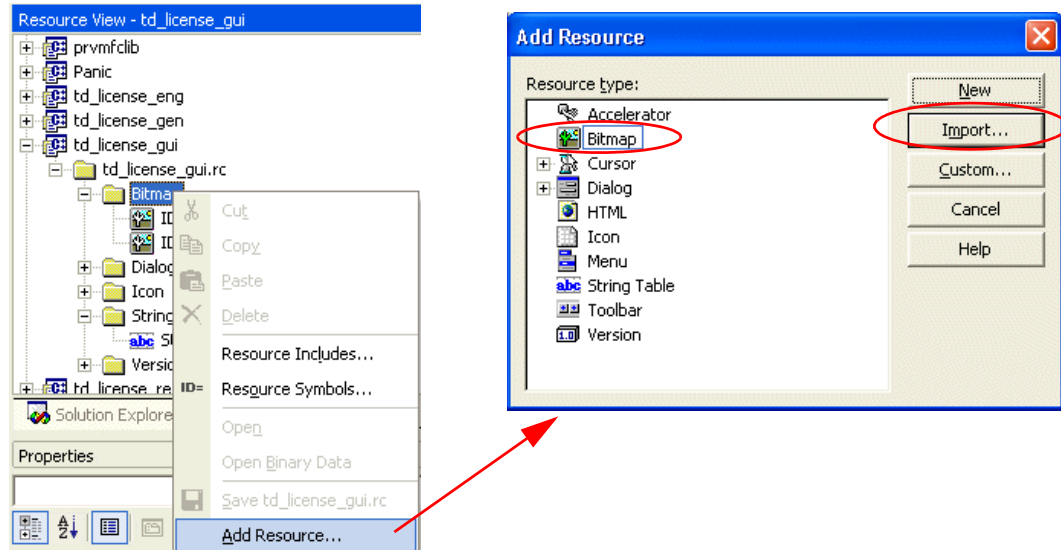
Step 4: Update the logo

The logo used by the default License UI files is called **rectangle.bmp** and is 177 x 442 pixels large. If your logo is larger, then you will need to modify the layout of the delivery dialog boxes in the project.

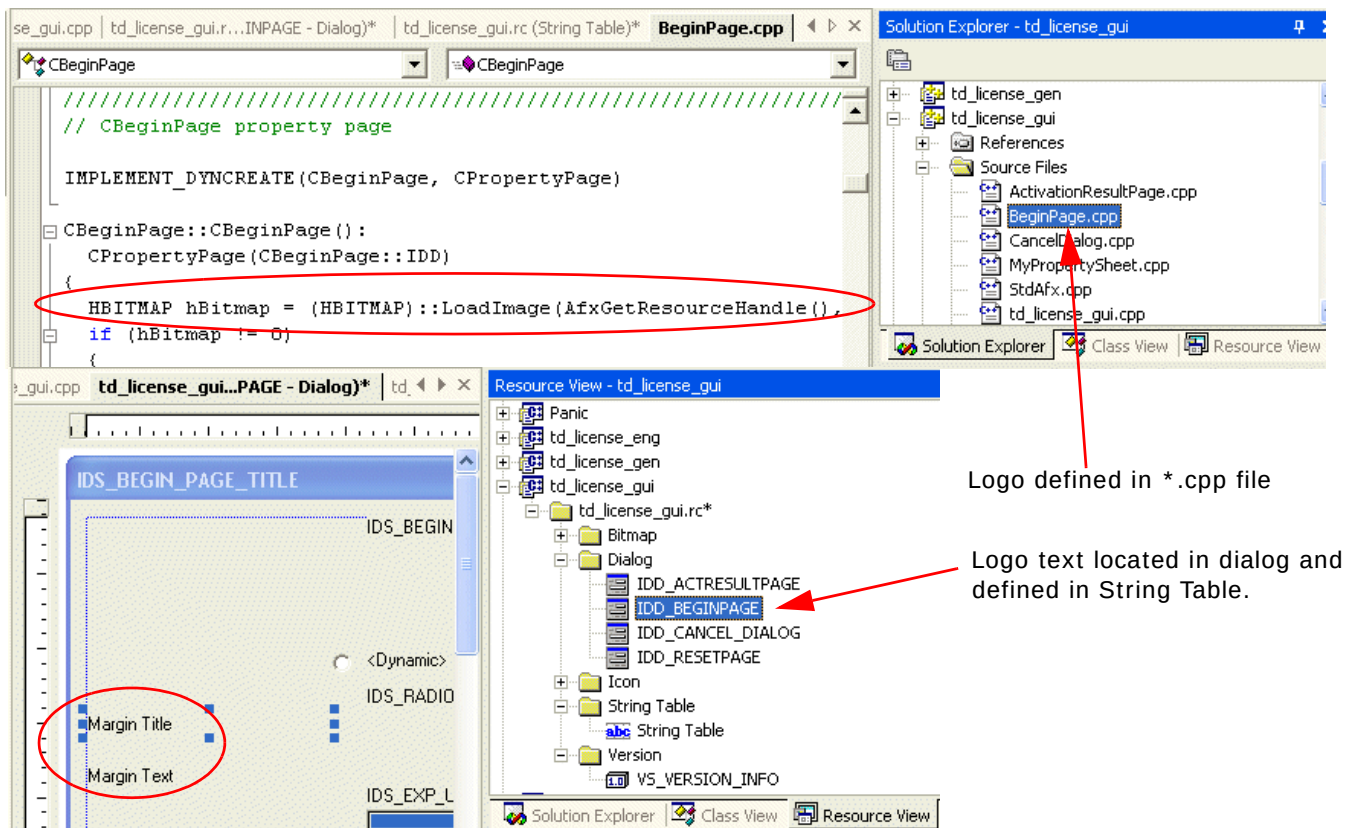
The dialogs are designed such that text displays on top of the logo. You may find it necessary to change the color of the text to be compatible with your logo, or simply remove it. Logo text is defined in the String Table.

To update the logo, it is easiest to rename your logo to "rectangle.bmp" and overwrite the existing logo in the **\res** folder where your project resides. For example, the rectangle.bmp for the td_license_gen project is in the following location: \client SDK\TD\td_license_gui\res.

If you want to use a differently named logo, then you must first import the new logo into your project. For example:



Then you must modify the source files of each of the dialogs to call your logo. For example:



Open all of the dialogs that use the logo and set them to use your custom logo. You are done updating the logo.

Step 5: Build your License Handler UI project

From the Build menu, build your License Handler UI DLL. Your file will depend on the output file specified for your project. See [To rename the output files](#).

Step 6: Add your new License Handler UI to the Usage Builder workspace

This is your last step. All HASP SL Framework files must be added to your Usage Builder Workspace via the Runtime Library tab. See the Usage Builder User Guide for details.

Localizing your client GUI files

Not only are the client GUI files 100% customizable, but they are easy to localize. For example, to localize your License Handler UI, do the following.

Send your License Handler GUI's String Table file to the your translator.

Open your `td_license_gui` Dialog files and adjust the License Handler UI dialog boxes as necessary to fit the translated text.

It's a good idea to follow the procedure [To rename the output files](#) to identify your language versions, such as "`my_td_license_gui_JP.dll`" for the Japanese-language version.

After you build localized client files you need to add them to your workspace via the Runtime Library tab in the Usage Builder.

Authoring DLLs for the HASP SL Framework

About

The Windows Reference Implementations are provided as MS Visual C++ .NET projects. However, you may write your own code for any Windows compiler. When doing so, please keep the following in mind:

The HASP SL Framework is an updateable runtime system, enabling you to modify the behavior or your license-managed software after it has been installed on your customers machine. To achieve this flexibility, your custom runtime libraries, such as a [License Handler](#), must support the defined HASP SL Framework interfaces.

Supporting the HASP SL runtime interfaces enables you to take advantage of the following attributes that the HASP SL Framework provides:

- Provide a way to be upgraded while being used by active clients.
- Enables your DLLs to be used by more than one application.

Runtime System Defined

We define a runtime system as a set of services used by multiple clients for an indefinite period of time. These services are presented as a set of interfaces (semantics + syntax) to these clients. These clients may join, leave and run concurrently with each other and may have multiple instances. The author of the runtime does not necessarily author these clients.

Runtime systems are useful because they unbind the implementation of services from their clients, allowing fixes to be made dynamically, after installation, without touching existing clients or notifying their owners. This late binding benefits both the client's users and authors.

The HASP SL Framework will evolve to include new features, bug fixes, etc. The evolution of the HASP SL Framework is asynchronous with the evolution of publisher-defined DLLs. The HASP SL Framework will always support existing interfaces.

A client invoking a service through an interface expects a certain defined external behavior. This behavior must remain constant as the runtime system evolves. The HASP SL Framework interfaces will be backwards compatible in all future releases.

HASP SL Framework Compliance

Guidelines for Authoring DLLs

We provide the following guidelines with respect to authoring DLLs, such as a License Handler, to be used with HASP SL Framework.

1. Use lower case when naming your DLL files.
2. Increment the Version number of your Library DLL files in the header files, in order that they can be used by HASP SL Licensing Framework, which is an updateable runtime system.
3. Use standard C linkage for all interface functions. Using C++ linkage can be problematic if different compilers use different name mangling strategies.
4. Don't use C++ types as parameters in interfaces. C++ objects that involve memory management will fail miserably across process boundaries.
5. Don't throw exceptions across DLL boundaries.
6. Don't rely on per-thread calls to the DLL initialization/finalization routines. Make the code in these trivial and reentrant.
7. Provide no static objects with non-trivial constructors (the actual invocation timing of the constructors is undefined).
8. Interfaces will expose functionality that requires global state. Make the lifetime of this state well defined as in a pair of {Initialize(), Finalize()} functions in the interface.

The calling convention of the API is fixed so that all the functions exported from a DLL can be called from any compiler.

The HASP SL Framework interface for License Handlers is called ILICENSE. Your License Handler must support a version of ILICENSE. All versions of ILICENSE will be supported by HASP SL, though new versions will be published. The Execution Environment communicates with your License Handler via the ILICENSE interface.

License Handler Interface: ILICENSE2

Notes

ILICENSE2 represents the second version of the ILICENSE interface. The HASP SL Licensing Framework supports both, as you can see from the code in the reference implementations.

This guide only documents the latest versions of published interfaces. For documentation about ILICENSE, refer to previous versions of the SDK Guide.

Change History

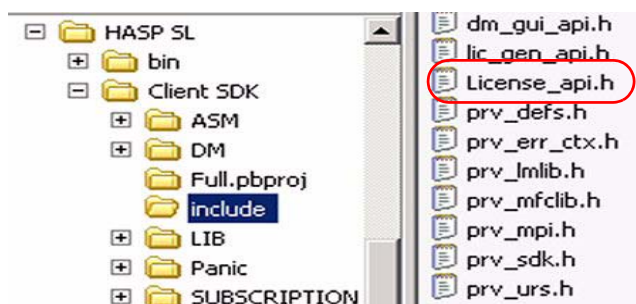
Interface Name	Changes from Previous Version
ILICENSE	Initial License Handler Interface.
ILICENSE2	Added GetLastError(), Enhanced Stop() to take boolean parameter, OkToRun() can be called with TRUE and FALSE, Replaced the build time function, GetUIDLLInterfaceName(), with a URS dictionary entry.

Description

The License Handler interface is mandatory. ILICENSE must be supported by your License Handler to be runtime compatible with the HASP SL Framework. The Execution Environment callbacks to your License Handler depend on the [Execution Environment: Interface and Services](#) your License Handler will use.

Lastly, the callbacks used between your License Handler and License Handler UI are 100% defined by you, as this communication resides outside of the HASP SL Framework. Because we provide the Product Activation License Handler and Product Activation Sample UI reference implementation for you to modify or extend, we will document the Product Activation Sample UI callbacks to the Product Activation License Handler.

The definition of ILICENSE2 can be found in license_api.h:



The interface is a structure with a list of function pointers named ILICENSE. License Handler must support the following interface to communicate with the Execution Environment. The Execution Environment will call the functions in a relative order, as described in the table below.

ILICENSE2 Interface Reference

Table of ILICENSE2 Calls

The HASP SL Execution Environment will call the functions in a relative order, as described in the table below.

Interface Function	Order Called	Description
Initialize()	1	Perform one-time initialization of the interface. This may be called more than once if more than one Protected File loads the License Handler.
Start()	2	Start an instance of your License Handler to handle a particular Protected File-License instance.
OkToRun()	3	When evaluating a license expression containing more than one valid license, the HASP SL Execution Environment will check the License Handler(s) associated with the licenses to see if the “unconditional” parameter of OkToRun() returns as TRUE, in which case, the Execution Environment will allow the Protected File to run without displaying the License Handler UI. Aladdin's License Handler reference implementations set “unconditional” to TRUE when a License is already activated.
InitialUserInteraction()	5	Display GUI to end user. This function is used to specify the GUI interaction when more than one valid License is available.
OkToRun()	6	This is called a second time only if the first OkToRun() returns FALSE for the “unconditional” parameter. Your License Handler must determine if it will allow the Execution Environment to execute the Protected File.
Charge()	7	Request from Execution Environment to charge usage of the Protected File-License session.
Tick()	Ongoing	Interval informing your License Handler that the Protected File-License session is being used.
Receive()	Optional	Function provided to support calls from a Protected File , which may be necessary when implementing feature-based licensing.
Stop()	2nd to last	Stops the instance of your License Handler. This is called when the Execution Environment shuts down the Protected File.
Finalize()	Last	Perform one-time shutdown of the interface. This may be called more than once if more than one Protected File loads your License Handler.
GetLastError()	Optional	Ask the execution environment to return an error context describing the last error the execution environment encountered.

Initialize()

```
bool_type (PRV_API * Initialize)(const IEE_SERVICES* services)
```

Description

Initializes the interface. The Execution Environment will not call functions prior to [Initialize\(\)](#).

Parameters Defined

Parameter	In / Out	Description
services	Input	Pointer to an Execution Environment service interface. See IEE_SERVICES below. This function may not block.

IEE_SERVICES

The Execution Environment provides the following services:

- [SV_DISCRETE_STORAGE2](#)
- [SV_CONTINUOUS_STORAGE2](#)
- [SV_LOCAL_HIDDEN_STORAGE2](#)
- [SV_LOCAL_EPHEMERAL_STORAGE](#)
- [SV_TEST_STORAGE](#)
- [SV_HARDWARE_FINGERPRINT](#)
- [SV_SOFTWARE_ACTIVATION3](#)
- [SV_SOFTWARE_ACTIVATION3_EX](#)
- [SV_DATE2](#)
- [SV_DESX](#)
- [SV_RANDOM](#)
- [SV_SYSTEMLOCK](#)
- [SV_URS](#)
- [SV_HASH](#)
- [SV_GUID](#)
- [SV_ERRORCONTEXTKEEPER](#)
- [SV_MPI](#)
- [SV_EC_INFO2](#)
- [SV_LOGGER](#)

Start()

```
bool_type (PRV_API * Start)(LicenseCallbackContext lcb_context,
    const unsigned char* license_key,
    const PRV_CHAR* secure_storage_selector,
    const PRV_CHAR* license_path,
    const PRV_CHAR* license_gui_dll_name,
    LicenseBehaviorModifier* behavior
    LicenseContext* lic_context)
```

Description

Create and start an instance of your License Handler. This function **may not** block.

[Start\(\)](#) returns TRUE if your License Handler starts, and FALSE if it cannot.

Parameters Defined

Parameter	In / Out	Description
lcb_context	In	Callback value for the Execution Environment context. The Execution Environment provides this value when your License Handler is started. See Start() . This value is used to call back to the Execution Environment.
license_key	In	DESX (16-byte) License key. See License key .
secure_storage_selector	In	Product identifier. The secure_storage_selector and the License name are used to define the address where data will be stored when your License Handler uses the Secure Storage Services Overview .
license_path	In	Full path to License that your License Handler will use to govern the Protected File. (UTF8)
license_gui_dll_name	In	Name of License Handler UI to be used with the specified Product and License. (UTF8)
behavior	In	0 = passive 1 = active The Product Activation License Handler uses this data as follows: Active means that usage will be charged when product is used. Passive means that usage will not be charged when product is used.
lic_context	Out	4-byte context ID for your License Handler to identify this invocation. This is unique ID used between the Start() and Stop() .

OkToRun()

```
bool_type (PRV_API * OkToRun)(LicenseContext lic_context,
    bool_type unconditional);
```

Description

Check if the license agrees to run the licensee. This function **may not** block.

Return TRUE to allow execution of Protected File. Return FALSE to prevent execution.

Parameters Defined

Parameter	In / Out	Description
lic_context	In	4-byte context ID for your License Handler to identify this invocation. See Start() .
unconditional	Out	Return TRUE if your License Handler agrees to run the Protected File unconditionally (without invoking UI or changing state). The HASP SL Reference Implementations use this parameter in the scenario when two or more valid licenses are installed, and one is activated, but the other is not. Rather than displaying the License Handler UI to get user input concerning the unactivated License, it will unconditionally run the Protected File.

InitialUserInteraction()

```
bool_type (PRV_API * InitialUserInteraction) (LicenseContext lic_context,
    bool_type more_licenses,
    UserInteractionResult* result);
```

Description

Perform the initial interaction with the user. This may be necessary when there is more than one valid License governing the Protected File, and your License Handler does not want to display the License Handler UI for each License. This function may block and display UI.

Return TRUE if function succeeds, and return FALSE otherwise. FALSE will stop execution of the Protected File.

Parameters Defined

Parameter	In / Out	Description
lic_context	In	4-byte context ID for your License Handler to identify this invocation. See Start() .
more_licenses	In	TRUE if there are more Licenses in the effective License Set for this Protected File.
result	Out	The result of the interaction. See UserInteractionResult values below.

UserInteractionResult

typedef enum tagUserInteractionResult

The result of a call to `ILICENSE::InitialUserInteraction()`.

Value	Description
LIC_UI_GO	Proceed, keep license in effective set. The Protected File's load will continue.
LIC_UI_STOP	Stop execution. The Protected File's load will be halted.
LIC_UI_IGNORE	Proceed, remove license in effective set. The Protected File's load will continue.

Charge()

```
bool_type (PRV_API * Charge)(LicenseContext lic_context)
```

Description

Charge usage of the Protected File to the License-Product pair. The Product Activation License Handler always charges usage whether or not a License's usage is actively enforced. This function **may not** block.

Return TRUE if License was charged. Return FALSE otherwise. Returning FALSE will stop execution of the Protected File.

Parameters Defined

Parameter	In / Out	Description
lic_context	In	4-byte context ID for your License Handler to identify this invocation. See Start() .

Tick()

```
bool_type (PRV_API * Tick)(LicenseContext lic_context, unsigned ms_interval)
```

Description

Time interval informing your License Handler that the Protected File-License session is in use. This function may not block. The return value is ignored.

Parameters Defined

Parameter	In / Out	Description
lic_context	Input	4-byte context ID for your License Handler to identify this invocation. See Start() .
ms_interval	Input	Millisecond interval. For example, "30,000" would be 30 seconds.

Receive()

```
bool_type (PRV_API * Receive)(LicenseContext lic_context,
    const unsigned char* in,
    unsigned in_length,
    unsigned char* out,
    unsigned* out_length);
```

Description

Receives the "command" from a publisher-defined [Protected File](#) calling the ISDK function [TransmitToLicenseDLL\(\)](#). This is used in a feature-based licensing scenario (see [License Data Implementation](#)), where application files send messages to the License Handler asking if they may run or not. License Handlers must be written such that `Receive()` supports every `TransmitToLicenseDLL()` message.

Parameters Defined

Parameter	In/Out	Description
lic_context	In	Context to identify instance.
in	In	Data received.
in_length	In	Length of received data.
out	Out	Data to send (allocated by caller).
out_length	Out	Length of data to send.

Stop()

```
bool_type (PRV_API * Stop)(LicenseContext lic_context, bool_type unload_dlls);
```

Stop and destroy an instance of a license state, unallocating all the memory since [Start\(\)](#) was called. This function may not block. The return value is ignored.

Parameters Defined

Parameter	In/Out	Description
lic_context	Input	Context to identify instance
unload_dlls	Input	Unload any loaded DLLs.

Finalize()

```
void (PRV_API * Finalize)()
```

Description

Finalizes an initialized interface. The Execution Environment will not call functions after calling [Finalize\(\)](#). This function **may not** block.

Parameters defined

None.

GetLastError()

```
const err_ctx* (PRV_API* GetLastError) ();
```

Return the last error context. Pertains to the last error resulting from calling functions in this interface or in any services obtained from this interface. Most of the functions returned by the Execution Environment are boolean, and, by design, no error data is passed. If one of your calls to the Execution Environment fails, then you can get error data by calling GetLastError().

See reference implementations for an example of how it is used.

Execution Environment: Interface and Services

IEE_SERVICES

```
typedef struct tagIEE_SERVICES{}
```

[IEE_SERVICES](#) is the interface supported by the Execution Environment that your License Handler calls in order to use the [Execution Environment Services](#). This interface is also defined in `license_api.h`.

Interface Function	Description
Initialize	Initialize an instance of the Execution Environment.
Finalize	Finalize the initialized instance.
GetServiceInterface	This allows the license DLL to obtain various services from the execution environment. See below.
GetLastError	Ask the execution environment to return an error context describing the last error the execution environment encountered.

Initialize()

```
bool_type (PRV_API * Initialize)();
```

Initialize the Interface. This function may not block.

Finalize()

```
void (PRV_API *Finalize)();
```

Finalize the interface. This function may not block.

GetServiceInterface()

```
bool_type (PRV_API* GetServiceInterface) (const PRV_CHAR* svc_name,  
    const void** svc_itf);
```

Get a service interface. See [Execution Environment Services](#). A service interface is a table of function pointers, usually defined as a structure with a name of SV_x. The pointers returned by this function are valid for the entire lifetime of the IEE_SERVICES interface.

Parameters Defined

Parameter	In / Out	Description
svc_name	In	Service name. See Execution Environment Services
svc_itf	Out	Pointer to resulting interface (no memory is allocated here)

GetLastError()

```
const err_ctx* (PRV_API* GetLastError) ();
```

Return the last error context. Pertains to the last error resulting from calling functions in this interface or in any services obtained from this interface. Most of the functions returned by the Execution Environment are boolean, and, by design, no error data is passed. If one of your calls to the Execution Environment fails, then you can get error data by calling GetLastError().

See reference implementations for an example of how it is used.

Execution Environment Services

The HASP SL Execution Environment provides the following API to extend the functionality of your License Handler. The prv_asm_lic.dll project is provided as a Visual C++ reference implementation for you to modify and reuse to meet your licensed managed software needs. prv_asm_lic.dll uses this API, thus providing examples of this API in practice.

Service	Description
SV_DISCRETE_STORAGE2	Discrete storage service
SV_CONTINUOUS_STORAGE2	Continuous storage service
SV_LOCAL_HIDDEN_STORAGE2	Locally hidden storage service
SV_LOCAL_EPHEMERAL_STORAGE	Local ephemeral storage service
SV_TEST_STORAGE	Test storage service
SV_HARDWARE_FINGERPRINT	Hardware fingerprinting service
SV_SOFTWARE_ACTIVATION3	Software activation service
SV_SOFTWARE_ACTIVATION3_EX	Software activation service (additional functions)
SV_DATE2	Date and time service
SV_DESX	Encryption/Decryption service
SV_DES	A different type of encryption option
SV_RANDOM	Randomizing service
SV_SYSTEMLOCK	Recursive, system wide named mutual exclusion lock service
SV_URS	Load publisher interface service
SV_HASH	Hash creating service
SV_GUID	GUID generating service.
SV_ERRORCONTEXTKEEPER	Service to track a per-thread error context value
SV_MPI	Memory protection interface service
SV_LOGGER	HASP SL client log service
SV_EC_INFO2	Obtain information about protected files

Secure Storage Services Overview

The Execution Environment provides four storage services for your license handlers or protected files that need to securely store data at runtime:

Table 3. Properties of Secure Storage Services

	Persistent	Immovable	Tamperproof	Secret	Hidden
SV_DISCRETE_STORAGE2	X		X	X	
SV_CONTINUOUS_STORAGE2	X	X	X	X	
SV_LOCAL_HIDDEN_STORAGE2	X		X	X	X
SV_LOCAL_EPHEMERAL_STORAGE			X	X	X
SV_TEST_STORAGE	X				

Table 4. Properties Defined

Persistent	Data whose lifetime is infinite after it is created and resides outside of a particular address space, such as on the local system's fixed disk. This data is not lost after the system shuts down. However it will be destroyed if the local system is destroyed.
Immovable	The storage and its data cannot be separated. Attempts to overwrite immovable data with previous data will result in failure.
Tamperproof	Data that cannot be modified to apply to an unintended product, protected file, or feature.
Secret	Data that cannot be read without holding a corresponding symmetric key.
Hidden	Data that is made difficult to find by Aladdin's hiding algorithm.

The Secure Storage services are supported by three functions: [PF_Put](#), [PF_Get](#), and [PF_Has](#).

SV_DISCRETE_STORAGE2

```
typedef struct tagSV_DISCRETE_STORAGE2
{
    PF_Put Put;
    PF_Get Get;
    PF_Has Has;
} SV_DISCRETE_STORAGE2;
```

Version 2 of Discrete Storage. Provides persistent, tamper proof, encrypted storage. Discrete storage is the most secure of the storage services and is recommended for any data that you want to secure on the customer's machine that is not continuous or needs to be hidden. For example, you would use discrete storage if you store fingerprint data to be analyzed at runtime for anti-piracy purposes.

For example, the HASP SL activation license handler stores the proof of activation in Discrete storage. See [Using the Windows Reference Implementations](#).

PF_Put

```
typedef bool_type (PRV_API* PF_Put)(StorageContext sc,
    unsigned item,
    const unsigned char * value,
    unsigned int value_length);
```

Parameters Defined

Parameter	In / Out	Description
sc	In	A Storage context can be a LicenseCallbackContext or an SDKContext. The Execution Environment provides the License Callback Context value when your License Handler is started. See Start() , and GetContext() .
item	In	Item number in storage. Bits 28-29 denotes scope. Aladdin reserves the last 2 bits 30-32 of an item identifier. (they must be 0).
value	In	Enter the beginning of the array of data to be stored.
value_length	In	Enter the length of the array of data (in bytes) to be stored.

Description

Put data in storage. Returns TRUE if computation succeeded, FALSE otherwise.

PF_Get

```
typedef bool_type (PRV_API* PF_Get)(StorageContext sc,
    unsigned item,
    unsigned char * const value,
    unsigned int * value_length);
```

Parameters Defined

Parameter	In/Out	Description
sc	In	A Storage context can be a LicenseCallbackContext or an SDKContext. The Execution Environment provides the License Callback Context value when your License Handler is started. See Start() , and GetContext() .
item	In	Item number in storage. Bits 28-29 denotes scope. Aladdin reserves the last 2 bits 30-32 of an item identifier. (they must be 0).
value	Out	Enter the beginning of the array of data to be retrieved from storage.
value_length	In/Out	Enter the length of the array of data (in bytes) to be retrieved from storage.

Description

Get data from storage. Returns TRUE if computation succeeded, FALSE otherwise.

PF_Has

```
typedef bool_type (PRV_API* PF_Has)(StorageContext sc,
    unsigned item,
    bool_type* has);
```

Parameters Defined

Parameter	In/Out	Description
sc	In	A Storage context can be a LicenseCallbackContext or an SDKContext. The Execution Environment provides the License Callback Context value when your License Handler is started. See Start() , and GetContext() .
item	In	Item number in storage. Bits 28-29 denotes scope. Aladdin reserves the last 2 bits 30-32 of an item identifier. (they must be 0).
has	Out	TRUE if item exists. FALSE if item does not exist.

Description

Query the existence of data from a storage service. Returns TRUE if computation succeeded, FALSE otherwise.

SV_CONTINUOUS_STORAGE2

```
typedef struct tagSV_CONTINUOUS_STORAGE2
{
    PF_Put Put;
    PF_Get Get;
    PF_Has Has;
} SV_CONTINUOUS_STORAGE2;
```

Version 2 of Continuous Storage. Designed to store continuous data, such as metering, time, or state data. Continuous storage prevents data from being rolled back to an older state. For example, the HASP SL activation license handler stores [License State](#) in continuous storage. Continuous storage is tamper-proof.

SV_LOCAL_HIDDEN_STORAGE2

```
typedef struct tagSV_LOCAL_HIDDEN_STORAGE2
{
    PF_Put Put;
    PF_Get Get;
    PF_Has Has;
} SV_LOCAL_HIDDEN_STORAGE2;
```

Version 2 of Hidden Storage. Designed to store data that needs to be hidden. Data that needs to be hidden is defined as information that is valuable to hacker if it is removed. For example, the HASP SL activation license handler stores the Trial Started bit in Local Hidden storage.

Local Hidden storage provides tamperproof storage protected by Aladdin's hiding algorithm. Ideally, you want to store the least amount of data in Hidden Storage, preferably only a single bit.

SV_LOCAL_EPHEMRAL_STORAGE

```
typedef struct tagSV_LOCAL_EPHEMRAL_STORAGE
{
    PF_Put Put;
    PF_Get Get;
    PF_Has Has;
} SV_LOCAL_EPHEMRAL_STORAGE;
```

Local Ephemeral Storage Service interface. Items in local ephemeral storage satisfy the tamperproof, secret, and hidden requirements, as they pertain to the memory image of a running process.

Local ephemeral is data whose lifetime is limited to the lifetime of its creator, and will be lost when the system is shut down. A [License Handler](#) may need local ephemeral storage for:

- Sharing data between itself and a protected file.
- Sharing data between different instances of itself protecting different protected files in a single address space.

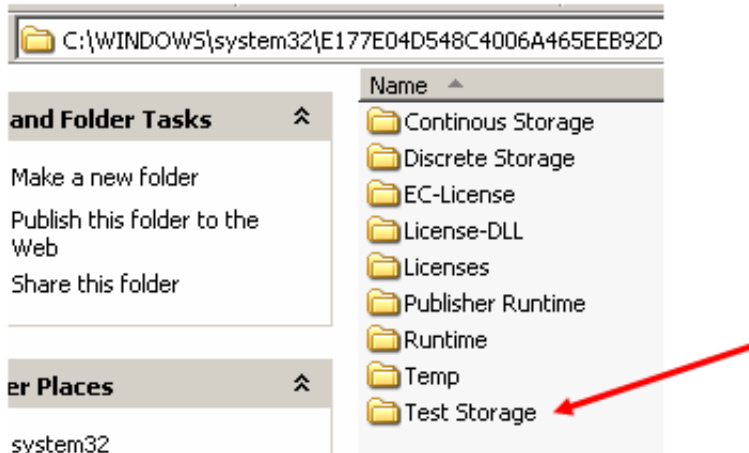
In addition to communicating with its license handler, a [Protected File](#) may use local ephemeral storage to securely store data needed for runtime operations, making it more resistant to runtime memory attacks.

Your protected product, for example, may transfer the product name to the license handler to display the 'product name' variable in the GUI.

SV_TEST_STORAGE

```
typedef struct tagSV_TEST_STORAGE
{
    PF_Put Put;
    PF_Get Get;
    PF_Has Has;
} SV_TEST_STORAGE;
```

This storage is for testing only. It is not secured (meaning that it is not encrypted and not tamper proof). Using it will create a Test Storage directory under the Hasp SL directory as shown below.



Example Code:

```

/*****

In this function we give an example for reading simple data (string) from a temporary
or Discrete storage and writing a string to a temporary or Discrete storage.
The API operations are:
1) PRV_SDK_LoadSDKInterface - connecting the SDK.
2) Initialize - Initializing the SDK.
3) GetContext - Getting the context of the SDK to have a handle for the Put/Get
operations.
4) PRV_SDK_LoadServiceInterface - Loading the service for our specific EC.
5) GetServiceInterface - Get the specific storage handle.
6) Put - Put the string in the storage (Write).
7) Get - Get the string from the storage and present it as output (Read).

*****/

// In this example we use Discrete Storage by default unless USE_TEST is defined.
#ifdef USE_TEST
#define STORAGE_TYPE    SV_TEST_STORAGE
#define STORAGE_NAME    SV_TEST_STORAGE_NAME
#else
#define STORAGE_TYPE    SV_DISCRETE_STORAGE2
#define STORAGE_NAME    SV_DISCRETE_STORAGE_NAME2
#endif

void CdiscobolDlg::OnBnClickedAnswer()
{
    char buf[MAX_PATH + 1]    = {0};
    char str[500]             = {0};

```

```

unsigned bufLen      = MAX_PATH;
ISDK* pSDK           = 0;
DWORD rc            = 0;
IEE_SERVICES* pSVC   = 0;
STORAGE_TYPE* pDS    = 0;
unsigned id          = (5 | SV_STORAGE_SCOPE_EC);
SDKContext ctx       = 0;

UpdateData(TRUE);

this->m_Log_str = "Path to file: ";

// Get real path in buf
rc = GetModuleFileName(0, buf, bufLen); // Return the size of the bytes read
this->m_Log_str += buf;

this->m_Log.SetWindowText(this->m_Log_str);
if (rc > 0) { // We read something...
    // Test: contacting the SDK
    rc = PRV_SDK_LoadSDKInterface(buf, &pSDK);
    if (rc) {
        this->m_Log_str += ".\r\nPRV_SDK_LoadSDKInterface returned\n\n";
        this->m_Log.SetWindowText(this->m_Log_str);
        // Initialize the SDK
        rc = pSDK->Initialize();
        if (rc) {
            this->m_Log_str += "Initialize returned successfull.\r\n";
            this->m_Log.SetWindowText(this->m_Log_str);
            // Obtain SDK context
            rc = pSDK->GetContext(buf, &ctx);
            if (rc) {
                // Test: contacting the Service Interface for our EC
                rc = PRV_SDK_LoadServiceInterface(&pSVC);
                if (rc) {
                    this->m_Log_str +=
                        "PRV_SDK_LoadServiceInterface returned\n\n";
                    this->m_Log.SetWindowText(this->m_Log_str);
                    // Get the specific storage handle
                    rc = pSVC->GetServiceInterface(STORAGE_NAME,
                                                    (const void*)&pDS);

                    if (rc) {
                        this->m_Log_str += "GetServiceInterface\n\n";
                        this->m_Log.SetWindowText(this->m_Log_str);
                        memset(buf, 0, sizeof(buf));
                        if (m_Is_Read == FALSE) { // We have a write event
                            // Copy the text from
                            // the edit box
                            strcpy(buf, m_to_store_str);
                            // Fill the storage with data
                            rc = pDS->Put(ctx,
                                          id,
                                          (unsigned char*)buf,
                                          sizeof(buf));

                            if (rc) {
                                this->m_Log_str += "Put returned\n\n";
                                this->m_Log.SetWindowText(this->m_Log_str);
                                MessageBox("String was saved");
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        successfully to the
        store", "Successfull
        Operation!",
        MB_ICONASTERISK);
    }
    else {
        Errors_handler("Error Put [Loader]");
    }
} //if
else { //if m_Is_Read = TRUE we have a read event
    // Clear the buffer so we will read the
    storage in to it :- )
    bufLen = sizeof(buf);
    // Get the storage
    rc = pDS->Get(ctx,
        id,
        (unsigned char*)buf,
        &bufLen);

    if (rc) {
        this->m_Log_str += "Get returned
        successfull.\r\n";
        this->m_Log.SetWindowText(this-
            >m_Log_str);
        // Set the storage text as output
        this->CThe_Answer.SetWindowText(buf);
        MessageBox("String was red
        successfully from store",
        "Successfull Operation!",
        MB_ICONASTERISK);
    } //if
    else {
        Errors_handler("Error Get [Loader]");
    }
} //else
} //if
else {
    Errors_handler("Error GetServiceInterface
    [Loader]");
}
} //if
else {
    Errors_handler("Error PRV_SDK_LoadServiceInterface
    [Loader]");
}
} //if
else {
    Errors_handler("Error GetContext [Loader]");
}
} //if
else {
    Errors_handler("Error Initializing SDK [Loader]");
}

// Finalizing...
pSDK->Finalize();
this->m_Log_str += "Finalize.\r\n";
this->m_Log.SetWindowText(this->m_Log_str);
// Unloading the SDK interface.
PRV_SDK_UnloadSDKInterface(pSDK);
this->m_Log_str += "PRV_SDK_UnloadSDKInterface.\r\n";
this->m_Log.SetWindowText(this->m_Log_str);
}

```

```
        else {  
            Errors_handler("Error loading SDK [Loader]");  
        }  
    }  
} //if  
m_Is_Read = TRUE;
```

SV_HARDWARE_FINGERPRINT

```
typedef struct tagSV_HARDWARE_FINGERPRINT
```

The Hardware Fingerprint service is provided to enable you to bind your product to a user's hardware. This provides anti-piracy in terms preventing your product from successfully running on a machine other than the one it was activated on.

Example Implementation of Hardware Binding

The following is an example of how you might implement hardware binding.

After successful product activation, your license handler would do the following. See also [How do Aladdin's License Models implement activation?](#)

1. The License Handler calls [GetHardwareFingerprint](#), specifying the [HWSL_NIC_MAC](#) and [HWSL_VOL_SERIAL](#) and [HWSL_CPU_SERIAL](#) selectors to generate fingerprint data.
2. The License Handler stores the fingerprint data in Discrete storage. See [Secure Storage Services Overview](#).
3. Each time an activated Product is initialized, the License Handler verifies the fingerprint data. It is considered invalid only if ALL of the fingerprint data is invalid (this will reduce support calls). Thus if any one of the MAC addresses, volume serial number, or CPU serial numbers in the fingerprint multi-string matches, then the fingerprint is valid.
4. If the fingerprint is valid, then the product runs without any interference from the HASP SL Framework.

If the fingerprint is invalid, then the Product Activation License Handler does two things:

- a. Change the state of this License-Product to "Dead." See [License State](#).
- b. Displays the License Handler UI to the user asking them to re-activate.

GetHardwareFingerprint

```
bool_type (PRV_API* GetHardwareFingerprint) (LicenseCallbackContext
lcb_context,
    HardwareSelector selector,
    char* value,
    unsigned* value_length);
```

Parameters Defined

Parameter	In / Out	Description
lcb_context	In	Callback value for the Execution Environment context. The Execution Environment provides this value when your License Handler is started. See Start() .
selector	In	Choose the type of fingerprint data you want to retrieve. See GetHardwareFingerprint Selector .
value	Out (optional)	Multi-string values, according to selector (memory allocated by caller).
value_length	In, Out	Length of multi-string. Max length is 2048 characters (unsigned).

Description

Retrieves a hardware fingerprint from the local machine based on the selector you have specified.

GetHardwareFingerprint Selector

```
typedef enum tagHardwareSelector
```

Hardware selectors to furnish GetHardwareFingerprint. Note that this data should be treated as transparent. This data is for binding purposes only and not for display.

Selector	Description
HWSL_NIC_MAC	Windows and Macintosh. Returns all of the NIC mac address(es) on local machine.
HWSL_NIC_IP	Windows and Macintosh. Returns all of the IP addresses defined for each NIC on local machine.
HWSL_VOL_SERIAL	Windows and Macintosh. Returns all of the volume serial number(s) on local machine.
HWSL_CPU_SERIAL	Windows only. Returns all of the CPU serial number(s) on local machine.
HWSL_BIOS_VERSION	Windows only. Returns version string of bios if possible.
HWSL_DRV_ID	Windows only. Returns a string for each physical drive on the system.

GetNextFingerprint

```
const char* (PRV_API* GetNextFingerprint)(const char* fp);
```

Return the next string in a multi-string fingerprint.

Parameters Defined

Parameter	In / Out	Description
fp	In	Where to start looking

[GetNextFingerprint](#) returns the next string in the fingerprint, or "0" if fp is the last string.

SV_SOFTWARE_ACTIVATION

```
typedef struct tagSV_SOFTWARE_ACTIVATION
```



This service has been left for backward compatibility only. New code should use [SV_SOFTWARE_ACTIVATION3](#) and its extension [SV_SOFTWARE_ACTIVATION3_EX](#).

This service is provided to support HASP SL Software Commerce Server, version 6.0 and later. In the Aladdin License Models that support activation, the License Handler uses this service to activate a product.

[SV_SOFTWARE_ACTIVATION](#) provides the following functions.

Function	Purpose
ActivatePerpetual	Runtime function used to activate a product.
ComputeChallenge	Runtime function used to compute the challenge code required by ActivatePerpetual .
CheckChallenge	Runtime function used to verify the challenge code returned by ActivatePerpetual .
GetOfferSecurity	This function is not used at runtime. The reference License Generators use this function to download a specific Offer's security data from the Activation Server Database in order to create an activation license in the Usage Builder.
GetOffers	This function is not used at runtime. The reference License Generator uses this function to download list of Offers available from an Activation Server Database in order to create a activation type licenses in the Usage Builder.

ActivatePerpetual

```
bool_type (PRV_API* ActivatePerpetual)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const char* activation_code,
    const char* distribution_code,
    const unsigned char* challenge,
    ActivationStatus* status,
    unsigned char* response);
```

Parameters Defined

Parameter	In / Out	Description
activation_url	In	URL of the Activation Server. Max length is 255 ANSI characters. The person creating the License defines this data in the Usage Builder at License creation time. The Usage Builder stores the Activation URL in the License file, where the License Handler gets it when the end user activates.
security	In	Security ID of Activation Server. The Usage Builder gets this data from the Activation Server Database using GetOfferSecurity , and stores it in the License file, where the License Handler gets it when the end user activates.
activation_code	In	Activation Code. This is entered into the License Handler UI by the end user.
distribution_code	In	Distribution Code. This is entered into the License Handler UI by the end user.
challenge	In	Challenge (8 bytes) derived from ComputeChallenge .
status	Out	Activation Status. See ActivatePerpetual Status Codes .
response	Out	Response from Activation Server (8 bytes).

[ActivatePerpetual](#) returns TRUE if the status from the server is SUCCESS.

[ActivatePerpetual](#) Status Codes

SA_STATUS_SUCCESS
SA_STATUS_SYSTEM_FAILURE
SA_STATUS_INVALID_ACT_CODE
SA_STATUS_DISTRIBUTION_DISABLED
SA_STATUS_ACT_CODE_DISABLED
SA_STATUS_MAX_ACT_EXCEEDED
SA_STATUS_INVALID_DIST_CODE
SA_STATUS_INVALID_PARAMETER
SA_STATUS_AUTHORIZATION_FAILED
SA_STATUS_UNKNOWN
SA_STATUS_INVALID_OFFER_USAGE

All messages are subject to input validation. All secure messages (SAClient and SASstorefront) are subject to authorization validation.

Description

Activate a product with the Admin Server.

ComputeChallenge

```
bool_type (PRV_API* ComputeChallenge)(const unsigned char* data,
    unsigned data_length,
    bool_type add_random,
    unsigned char* challenge);
```

Parameters Defined

Parameter	In / Out	Description
data	In	Base data to use to compute the challenge
data_length	In	Length of the data
add_random	In	TRUE if random data should be added to challenge
challenge	Out	Challenge (8 bytes)

Description

The ComputeChallenge function is used for three different purposes by Aladdin's Product Activation model reference implementation.

1. Activation

The Activation Server requires a challenge code in order to activate a product. The Product Activation License Handler uses [ComputeChallenge](#) to generate this challenge code. The *data* parameter is obtained from the [SV_HARDWARE_FINGERPRINT](#) service by the License Handler. The *add_random* parameter is set to FALSE, so that the same challenge code may be generated each time, when given the same input data.

2. Trial Reset

In this case, the *add_random* parameter is set to TRUE, so that the Trial Reset challenge and corresponding Trial Reset response are different each time, ensuring that a Trial Reset response cannot be used more than once.

3. Unlocking an already activated License

If the state of a License goes to Dead for some reason, the end user must send an unlock request to the Activation Server via phone or internet in order to unlock this state. The *add_random* parameter is set to TRUE, so that the Unlock challenge and corresponding Unlock response are different each time. See also [License State](#).

CheckChallenge

```
bool_type (PRV_API* CheckChallenge)(const unsigned char* activation_key,
    const unsigned char* challenge,
    const unsigned char* response);
```

Parameters Defined

Parameter	In / Out	Description
activation_key	In	DES activation key (8 bytes)
challenge	In	Challenge (8 bytes)
response	Out	Response (8 bytes)

Description

Check if a response is valid for a given challenge. The Product Activation License Handler reference implementation uses this function is used to validate the Activation Server by verifying the response returned by [ActivatePerpetual](#).

GetOfferSecurity

```
bool_type (PRV_API* GetOfferSecurity)(const char* admin_url,
    const SA_OFFER_INFO* offer_info,
    SA_OFFER_SECURITY* security);
```

Parameters Defined

Parameter	In / Out	Description
admin_url	In	Admin Server URL. Max length is 255 ANSI characters.
offer_info	In	Offer to get information about. See SA_OFFER_INFO .
security	Out/In	Security information of offer. See SA_OFFER_SECURITY .

Description

[GetOfferSecurity](#) is used only at build time by the License Generator when a License is created. The Usage Builder places the [SA_OFFER_SECURITY](#) data returned by this function into the License file, where the Product Activation License Handler gets it at runtime when the end user requests product activation.

GetOffers

```
bool_type (PRV_API* GetOffers)(const char* admin_url,
    SA_OFFER_INFO* offer_info,
    unsigned* num_offers);
```

Parameters Defined

Parameter	In / Out	Description
admin_url	In	Administrator Server URL. Max length is 255 ANSI characters. See the Usage Builder online Help for more info.
offer_info	Out	Offer to get information about. See SA_OFFER_INFO .
num_offers	In, Out	List of offers available at specified admin_url.

Description

[GetOffers](#) is used only at build time by the License Generator when a License is created. The Usage Builder displays the list of Offers returned by this function to the user who selects the appropriate Offer for the License being created.

SV_SOFTWARE_ACTIVATION_EX

typedef struct tagSV_SOFTWARE_ACTIVATION_EX



This service has been left for backward compatability only. New code should use [SV_SOFTWARE_ACTIVATION3](#) and its extension [SV_SOFTWARE_ACTIVATION3_EX](#).

This service extends the functionality provided by [SV_SOFTWARE_ACTIVATION](#), for the purpose of supporting features provided by the HASP SL Commerce Server, version 6.2, such as publishers and offer usage. Note that this service is not a replacement for [SV_SOFTWARE_ACTIVATION](#).

[SV_SOFTWARE_ACTIVATION_EX](#) provides the following functions.

Function	Purpose
GetPublishers	Used by License Generator to display a list of publishers in Usage Builder.
ActivatePerpetual	Activate to enable perpetual usage. Used to activate on a version 6.2 or later HASP SL Commerce Server.
RecordTrialStart	Records the start of a trial on a version 6.2 or later HASP SL Commerce Server using the URL provided in the License.
ActivateTrial	Activate to enable trial start. Used to activate on a version 6.2 or later HASP SL Commerce Server.
GetServerVersion	Used at build time by the License Generator to determine the version of the HASP SL Commerce Server.
ComputeChallenge	Runtime function used to compute the challenge code required by ActivatePerpetual and ActivateTrial .

GetPublishers

```
bool_type (PRV_API* GetPublishers)(const char* admin_url,
    SA_PUBLISHER_INFO* publisher_info,
    unsigned* num_publishers);
```

Parameters Defined

Parameter	In / Out	Description
admin_url	In	Admin Server URL
publisher_info	Out	Publishers will be placed here
num_publisher	In, Out	Number of offers in publisher_info, out: required size of publisher_info

Description

Get all of the publishers defined on a HASP SL Commerce Server. This function is used by License Generator to display a list of publishers in Usage Builder.

ActivatePerpetual

```
bool_type (PRV_API* ActivatePerpetual)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const char* activation_code,
    const char* distribution_code,
    ActivationStatus* status);
```

Parameters Defined

Parameter	In / Out	Description
activation_url	In	URL of activation server
security	In	Offer Security
activation_code	In	Activation code
distribution_code	In	Distribution code
status	Out	Activation status (one of ActivationStatus)

Description

Perpetually activate the “activation code + distribution code” pair using the Admin Server 6.2 semantics. This function returns TRUE if communication to server succeeded.

RecordTrialStart

```
bool_type (PRV_API* RecordTrialStart)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const SA_OFFER_INFO* offer,
    const PRV_CHAR* campaign_code);
```

Parameters Defined

Parameter	In / Out	Description
activation_url	In	URL of activation server
security	In	Offer Security
offer	In	Offer to record trial start of. Offer on server must have SA_OFFER_USAGE_TRIAL in it's usage mask
campaign_code	In	Campaign code to use. If the value is "", the campaign_code will not be sent to the server.

Description

Record the start of a trial for a given offer. This function returns TRUE if communication to server succeeded.

ActivateTrial

```
bool_type (PRV_API* ActivateTrial)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const char* activation_code,
    const char* distribution_code,
    ActivationStatus* status);
```

Parameters Defined

Parameter	In/Out	Description
activation_url	In	URL of activation server
security	In	Offer Security
activation_code	In	Activation code
distribution_code	In	Distribution code
status	Out	Activation status (one of ActivationStatus)

Description

Activate the trial using with “activation code + distribution code” pair using Admin Server 6.2 semantics. This function returns TRUE if communication to server succeeded.

GetServerVersion

```
bool_type (PRV_API* GetServerVersion)(const char* admin_url,
    unsigned* server_contract_version);
```

Parameters Defined

Parameter	In/Out	Description
admin_url	In	Admin Server URL.
version	Out	Server contract version.

Description

Get HASP SL Commerce Server version. This function is used by the License Generator at build time.

ComputeChallenge

```
bool_type (PRV_API* ComputeChallenge)(bool_type add_random,
    unsigned char* challenge);
```

Parameters Defined

Parameter	In/Out	Description
add_random	In	TRUE if random data should be added to challenge
challenge	Out	Challenge (8 bytes)

Description

Compute a Challenge for the Extended Activation Service.

NOTE: Extended Activation Service challenges always have the same data: The HASP SL install GUID.

SV_SOFTWARE_ACTIVATION3

typedef struct tagSV_SOFTWARE_ACTIVATION3

SV_SOFTWARE_ACTIVATION3 provides the following functions.

Function	Purpose
ActivatePerpetual	Activate to enable perpetual usage.
ComputeChallenge	Runtime function used to compute the challenge code required by ActivatePerpetual and ActivateTrial .
CheckChallenge	Runtime function used to verify the challenge code returned by ActivatePerpetual .
GetOfferSecurity	This function is not used at runtime. The reference License Generators use this function to download a specific Offer's security data from the Activation Server Database in order to create an activation license in the HASP Usage Builder.
GetOffers	This function is not used at runtime. The reference License Generator uses this function to download list of Offers available from an Activation Server Database in order to create a activation type licenses in the HASP Usage Builder.
RecordTrialStart	Records the start of a trial on a version 6.4 or later HASP SL Commerce Server using the URL provided in the License.
ActivateTrial	Activate to enable trial start. Used to activate on a version 6.4 or later HASP SL Commerce Server.
GetServerVersion	Used at build time by the License Generator to determine the version of the HASP SL Commerce Server.
RegisterUser	Register client information.
InitRegData	Initiates a data registration sequence.
AddRegData	Accumulates registration data.
CommitRegistration	Finalizes the add_reg_data sequence and commits on the server.
CustomerDataHas	Query if customer data exists.
CustomerDataGet	Get customer data.
CustomerDataPut	Put customer data.

How do Aladdin's License Models implement activation?

How is a License activated? Activation is an implementation choice. The Activation service provided by the Execution Environment service directly supports the HASP SL Software Commerce Server.

If you import custom Activation Codes into the HASP SL Server Database, then you must modify the code in the reference implementation License Handler UI to handle and display the format of your Activation Codes.

Build Time: Create an Aladdin Activation Type License

1. At License creation time, create a new license from one of Aladdin's activation type license generators. Then choose a publisher key file, and the Usage Builder displays the list of offers (using [GetOffers](#)).
2. Choose an Offer from the list. The Usage Builder then procures the following data from the HASP SL Admin Server, and places it in the License (using [GetOfferSecurity](#)):
 - a. Offer key
 - b. Activation Server Communication key
 - c. Security ID

See [License](#) for a list of all the items placed into the License. See also the Usage Builder User Guide for License creation details.

Runtime: Product Activation

1. The License Handler displays the License Handler GUI to the user, whereby the user chooses to activate, and enters an Activation Code.
2. The License Handler submits Offer data in the License, the Activation Code and Distribution Code from the end user using [ActivatePerpetual](#).

Activation Requests will be the same for the same product on the same machine.
3. The Activation Server returns a Success status and a response. See [ActivatePerpetual Status Codes](#). The License Handler submits the response to [CheckChallenge](#) to verify it.
4. The License Handler then changes the License State to "Activated." See [License State](#).

ActivatePerpetual

```
bool_type (PRV_API* ActivatePerpetual)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const char* activation_code,
    const char* distribution_code,
    const unsigned char* challenge,
    ActivationStatus* status,
    unsigned char* response,
    char* license_data);
```

Parameters Defined

Parameter	In / Out	Description
activation_url	In	URL of the Activation Server. Max length is 255 ANSI characters. The person creating the License defines this data in the Usage Builder at License creation time. The Usage Builder stores the Activation URL in the License file, where the License Handler gets it when the end user activates.
security	In	Security ID of Activation Server. The Usage Builder gets this data from the Activation Server Database using GetOfferSecurity , and stores it in the License file, where the License Handler gets it when the end user activates.
activation_code	In	Activation Code. This is entered into the License Handler UI by the end user.
distribution_code	In	Distribution Code. This is entered into the License Handler UI by the end user.
challenge	In	Challenge (8 bytes) derived from ComputeChallenge .
status	Out	Activation Status. See ActivatePerpetual Status Codes .
response	Out	Response from Activation Server (8 bytes).
license_data	Out	License data (1024 bytes).

[ActivatePerpetual](#) returns TRUE if the status from the server is SUCCESS.

Description

Activate a product with the HASP SL Activation Server, version 6.4.

ComputeChallenge

```
bool_type (PRV_API* ComputeChallenge)
    (bool_type add_random,
     unsigned char* challenge);
```

Parameters Defined

Parameter	In / Out	Description
add_random	In	TRUE if random data should be added to challenge
challenge	Out	Challenge (8 bytes)

Description

The ComputeChallenge function is used for three different purposes by Aladdin's Product Activation model reference implementation.

1. Activation

The Activation Server requires a challenge code in order to activate a product. The Product Activation License Handler uses [ComputeChallenge](#) to generate this challenge code. The *data* parameter is obtained from the [SV_HARDWARE_FINGERPRINT](#) service by

the License Handler. The `add_random` parameter is set to FALSE, so that the same challenge code may be generated each time, when given the same input data.

2. Trial Reset

In this case, the `add_random` parameter is set to TRUE, so that the Trial Reset challenge and corresponding Trial Reset response are different each time, ensuring that a Trial Reset response cannot be used more than once.

3. Unlocking an already activated License

If the state of a License goes to Dead for some reason, the end user must send an unlock request to the Activation Server via phone or internet in order to unlock this state. The `add_random` parameter is set to TRUE, so that the Unlock challenge and corresponding Unlock response are different each time. See also [License State](#).

CheckChallenge

```
bool_type (PRV_API* CheckChallenge)(const unsigned char* activation_key,
    const unsigned char* challenge,
    const unsigned char* response);
```

Parameters Defined

Parameter	In/Out	Description
activation_key	In	DES activation key (8 bytes)
challenge	In	Challenge (8 bytes)
response	In	Response (8 bytes)

Description

Check if a response is valid for a given challenge. The Product Activation License Handler reference implementation uses this function is used to validate the Activation Server by verifying the response returned by [ActivatePerpetual](#).

GetOfferSecurity

```
bool_type (PRV_API* GetOfferSecurity)(const char* publisher_url,
    const char* publisher_keyfile,
    const SA_OFFER_INFO* offer_info,
    SA_OFFER_SECURITY* security);
```

Parameters Defined

Parameter	In/Out	Description
publisher_url	In	Publisher Server URL.
publisher_keyfile	In	Path to the keyfile.
offer_info	In	Offer to get information about. See SA_OFFER_INFO .
security	Out	Security information of offer. See SA_OFFER_SECURITY .

Description

[GetOfferSecurity](#) is used only at build time by the License Generator when a License is created. The Usage Builder places the [SA_OFFER_SECURITY](#) data returned by this function into the License file, where the Product Activation License Handler gets it at runtime when the end user requests product activation.

Data Structures Defined

SA_OFFER_SECURITY

```
typedef struct tagSA_OFFER_SECURITY
```

This data structure is comprised of the following:

Parameter	Description
unsigned csize	Structure size
unsigned security_id	Security ID
char communication_key	Communication key. Encoded Activation keys are a maximum of 1024 characters (good for key lengths up to 2048 bits).
unsigned char activation_key	Offer key. DES activation key. Must be 8 bytes.
bool_type has_challenge_key	Validity bit for challenge key.
unsigned char challenge_key[65]	Phone activation "License Data" challenge key.

SA_OFFER_INFO

```
typedef struct tagSA_OFFER_INFO
```

This data structure is comprised of the following:

Parameter	Description
unsigned csize	Structure size
unsigned offer_id	Offer ID. Limited to 255 ANSI chars.
char description	Offer Name. Maximum length = 255 ANSI characters.
unsigned publisher_id	Publisher ID. Note that this is unique per server.
SA_OFFER_USAGE usage	Offer Usage bitmask. See SA_OFFER_USAGE .

SA_OFFER_USAGE

```
typedef enum tagSA_OFFER_USAGE
```

This data structure defines the possible usage of an offer. Offer usage is defined at time of creation. See the *Commerce Server Administrator User Guide* for details.

Parameter	Description
SA_OFFER_USAGE_TRIAL	Offer may be used for trial activation, whereby trials must be activated prior to use.
SA_OFFER_USAGE_ACTIVATION	Offer may be used for perpetual activation, whereby activation is required to transition a product into perpetual use.

GetOffers

```
bool_type (PRV_API* GetOffers)(const char* publisher_url,
    const char* publisher_keyfile,
    SA_OFFER_INFO* offer_info,
    unsigned* num_offers);
```



This function can fulfill [SA_OFFER_INFO](#) or SA_OFFER_INFO2 requests depending on the cbsize field of each structure.

Parameters Defined

Parameter	In / Out	Description
publisher_url	In	Publisher Server URL.
publisher_keyfile	In	Path to the keyfile.
offer_info	Out	Offers will be placed here. Place NULL to get required size.
num_offers	In, Out	In: Number of offers in offer_info

Description

[GetOffers](#) is used only at build time by the License Generator when a License is created. The Usage Builder displays the list of Offers returned by this function to the user who selects the appropriate Offer for the License being created.

RecordTrialStart

```
bool_type (PRV_API* RecordTrialStart)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const SA_OFFER_INFO* offer,
    const PRV_CHAR* campaign_code);
```

Parameters Defined

Parameter	In/Out	Description
activation_url	In	URL of activation server
security	In	Offer Security
offer	In	Offer to record trial start of. Offer on server must have SA_OFFER_USAGE_TRIAL in it's usage mask
campaign_code	In	Campaign code to use. If the value is "", the campaign code will not be sent to the server.

[RecordTrialStart](#) returns TRUE if communication to the server succeeded.

Description

Record the start of a trial for a given offer. This function returns TRUE if communication to server succeeded.

ActivateTrial

```
bool_type (PRV_API* ActivateTrial)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const char* activation_code,
    const char* distribution_code,
    const unsigned char* challenge,
    ActivationStatus* status,
    unsigned char* response);
```

Parameters Defined

Parameter	In/Out	Description
activation_url	In	URL of activation server
security	In	Offer Security
activation_code	In	Activation code
distribution_code	In	Distribution code
status	Out	Activation status (one of ActivationStatus)
response	Out	Response from activation server (8 bytes)

Description

Activate the trial of a pair (activation, distribution). This function returns TRUE if communication to server succeeded.

GetServerVersion

```
bool_type (PRV_API* GetServerVersion)(const char* admin_url,
    unsigned* server_contract_version);
```

Parameters Defined

Parameter	In/Out	Description
admin_url	In	Admin Server URL.
version	Out	Server contract version.

Description

Get HASP SL Commerce Server version. This function is used by the License Generator at build time.

RegisterUser

```
bool_type (PRV_API* RegisterUser)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const SA_OFFER_INFO* offer_info,
    const char* firstName,
    const char* lastName,
    const char* emailAddress)
```



Register_user function is a syntactic sugar of the following function sequence:

```
RegContextHandler h;
InitRegData(activation_url, security, &h);
add_reg_data(COLNAME_FIRST_NAME, firstName, h);
add_reg_data(COLNAME_LAST_NAME, lastName, h);
add_reg_data(COLNAME_EMAIL_ADDRESS, emailAddress, h);
commit_registration(o, h);
```

Parameters Defined

Parameter	In/Out	Description
activation_url	In	URL of the activation server
security	In	Offer security context
offer	In	Offer for which to submit registration data
firtsname	In	
lastname	In	
emailaddress	In	

Description

Register client information

InitRegData

```
bool_type (PRV_API* InitRegData)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    RegContextHandler* hRegCtx)
```

Parameters Defined

Parameter	In / Out	Description
activation_url	In	URL for activation server
security	In	Offer security context
hRegCtx	Out	Pointer to handler to registration context

Description

Initiates a data registration sequence.

AddRegData

```
bool_type (PRV_API* AddRegData)(const char* strKey,
    const char* strVal,
    RegContextHandler hRegCtx)
```

Parameters Defined

Parameter	In / Out	Description
strKey	In	Registration data key
strVal	In	Registration data value
hRegCtx	In	Handler to registration context



The character “[” may not be used in the strKey or strVal parameters. Using this value may cause problems in your registration data reports.

Description

Accumulates registration data.

CommitRegistration

```
bool_type (PRV_API* CommitRegistration)(const SA_OFFER_INFO* offer_info,
    RegContextHandler hRegCtx)
```

Parameters Defined

Parameter	In / Out	Description
offer_info	In	Offer for which to record data
hRegCtx	In	Handler registration context

Description

Finalizes the add_reg_data sequence and commits on the server.

CustomerDataHas

```
bool_type (PRV_API* CustomerDataHas)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const char* activation_code,
    const char* distribution_code,
    const char* key,
    bool_type* has)
```

Parameters Defined

Parameter	In / Out	Description
activation_url	In	URL for activation server
security	In	Offer security context
activation_code	In	Activation code
distribution_code	In	Distribution code
key	In	Key name (description)
has	Out	Put value here

Description

Query if customer data exists.

CustomerDataGet

```
bool_type (PRV_API* CustomerDataGet)(const char* activation_url,
    const SA_OFFER_SECURITY* security,
    const char* activation_code,
    const char* distribution_code,
    const char* key,
    char* const value,
    unsigned* const value_len)
```

Parameters Defined

Parameter	In / Out	Description
activation_url	In	URL for activation server
security	In	Offer security context
activation_code	In	Activation code
distribution_code	In	Distribution code
key	In	Key name (description)
value	Out	Put value here (ANSI string)
value_len	In, Out	In: Length of value, Out: Expected length

Description

Get customer data.

CustomerDataPut

```
bool_type (PRV_API* CustomerDataPut)(const char* activation_url,  
    const SA_OFFER_SECURITY* security,  
    const char* activation_code,  
    const char* distribution_code,  
    const char* key,  
    const char* value)
```

Parameters Defined

Parameter	In / Out	Description
activation_url	In	URL for activation server
security	In	Offer security context
activation_code	In	Activation code
distribution_code	In	Distribution code
key	In	Key name (description)
value	Out	Put value here (ANSI string)

Description

Put customer data.

SV_SOFTWARE_ACTIVATION3_EX

typedef struct tagSV_SOFTWARE_ACTIVATION3_EX

[SV_SOFTWARE_ACTIVATION3_EX](#) provides the following functions.

Function	Purpose
AddRegDataW	Accumulates registration data - the Unicode version (Win32 only).
CheckPhoneChallenge	Check if a response is valid for a given challenge and license data.

AddRegDataW

```
bool_type (PRV_API* AddRegDataW)( const U_CHAR* wstrKey,
    const U_CHAR* wstrVal,
    RegContextHandler hRegCtx);
```

Parameters Defined

Parameter	In/Out	Description
wstrKey	In	Registration data key
wstrVal	In	Registration data value
hRegCtx	In	Handler to registration contex

Description

Accumulates registration data - the Unicode version.

CheckPhoneChallenge

```
bool_type (PRV_API* CheckPhoneChallenge)
( const unsigned char* challenge,
  const unsigned char* response,
  const unsigned char* license_data,
  const unsigned char* chalenge_key);
```

Parameters Defined

Parameter	In/Out	Description
challenge	In	Challenge code (8 bytes)
response	In	Expected response code (8 bytes)
license_data	In	License data
chalenge_key	In	DES activation key (8 bytes)

Description

Check if a response is valid for a given challenge and license data.

SV_DATE2

```
typedef struct tagSV_DATE2
```

This service provides date and time values so that your code does not have to support date time code. The Product Activation License Handler uses this service to calculate time limits for trial licenses. it replaces SV_DATE.

[SV_DATE2](#) provides the following functions.

Function	Purpose
GetDate	Obtains the date from the BIOS of the local machine.
GetBestDate	The best available date is searched for in this order: Internet, LAN, then local.
Add	Add values to a date.
Compare	Compares 2 dates
Convert	Converts a date from a different time-zone format

GetDate

```
bool_type (PRV_API* GetDate)(DATE_TIMEZONE tz, PRV_DATE* date);
```

Parameters Defined

Parameter	In / Out	Description
tz	In	The requested time-zone.
date	Out	Date derived from the BIOS of the local machine.

Description

Obtains the date from the BIOS of the local machine.

GetBestDate

```
bool_type (PRV_API* GetBestDate)(DATE_TIMEZONE tz, PRV_DATE* date);
```

Parameters Defined

Parameter	In / Out	Description
tz	In	The requested time-zone.
date	Out	Derives the most accurate date.

Description

The best available date is searched for in this order: Internet, LAN, then local. The [GetBestDate](#) will return the first date it finds.

Add

```
bool_type (PRV_API* Add)(const PRV_DATE* src,  
    const PRV_DATE* v,  
    PRV_DATE* dst);
```

Parameters Defined

Parameter	In / Out	Description
src	In	Date
v	In	The values to add. The following values are permissible: day, hour, minute and seconds. The other values must be set to 0.
dst	Out	The newly computed date.

Description

Add values to a date.

Compare

```
bool_type (PRV_API* Compare)(const PRV_DATE* d1,  
    const PRV_DATE* d2,  
    int* result);
```

Parameters Defined

Parameter	In / Out	Description
d1	In	First date
d2	In	Second date
result	Out	0 if d1 == d2, 1 if d1 > d2, -1 if d1 < d2

Description

Compares two given dates.

Convert

```
bool_type (PRV_API* Convert)(const PRV_DATE* src_date,  
    DATE_TIMEZONE src_tz,  
    DATE_TIMEZONE dst_tz,  
    PRV_DATE* dst_date);
```

Parameters Defined

Parameter	In / Out	Description
src_date	In	Source date.
src_tz	In	Source time-zone.
dst_tz	In	Destination time-zone.
dst_date	Out	Converted date.

Description

Convert a date from a different time-zone format.

SV_DESX

```
typedef struct tagSV_DESX
```

This service provides a single encryption/decryption service to be used in conjunction with the [License key](#) to encrypt whatever data you don't want your user's to see. The Execution Environment passes the License key to your License Handler with the [Start\(\)](#) function.

Process()

```
bool_type (PRV_API* Process)(const unsigned char* key,  
    unsigned char* data,  
    unsigned data_len,  
    bool_type encrypt);
```

Parameters Defined

Parameter	In / Out	Description
key	In	DESX key (16 bytes) to use for encryption/decryption. The Product Activation License Handler uses the License key .
data	In, Out	Data to process, output buffer (must be aligned on 8-byte boundary)
data_len	In	Length of data you want to encrypt. This function does not limit the length.
encrypt	In	TRUE if encryption desired FALSE if decryption desired

Description

Encrypt or Decrypt a buffer using a DESX key.

SV_DES

```
typedef struct tagSV_DES
```

This service has been added to provide another encryption/decryption option using DES.

Process()

```
bool_type (PRV_API* Process)(const unsigned char* key,  
    unsigned char* data,  
    unsigned data_len,  
    bool_type encrypt);
```

Parameters Defined

Parameter	In / Out	Description
key	In	DES key (8 bytes) to use for encryption/decryption.
data	In, Out	Data to process, output buffer (must be aligned on 8-byte boundary)
data_len	In	Length of data you want to encrypt. This function does not limit the length.
encrypt	In	TRUE if encryption desired FALSE if decryption desired

Description

Encrypt or Decrypt a buffer using a DES key.

SV_HASH

```
typedef struct tagSV_HASH  
Hashing Services
```

Create()

```
bool_type (PRV_API* Create)(HashAlgorithm a,  
    HashContext* ctx);
```

Parameters Defined

Parameter	In / Out	Description
a	In	Algorithm to choose, one of HashAlgorithm enumeration
ctx	In	Hash context

Description

Create an hash context

Supported Hash Algorithms

HashAlgorithm

```
typedef enum tagHashAlgorithm  
SV_HASH_MD5 = MD5 hash
```

Update()

```
bool_type (PRV_API* Update)(HashContext ctx,  
    const unsigned char* data,  
    unsigned data_len);
```

Parameters Defined

Parameter	In / Out	Description
ctx	Out	Hash context
data	In	Data to update
data_len	In	Length of data

Description

Update the hash

Close()

```
bool_type (PRV_API* Close)(HashContext ctx,  
    unsigned char* hash);
```

Parameters Defined

Parameter	In / Out	Description
ctx	In	Hash context to use
hash	Out	Resulting hash

Description

Close the hash and return the result. The HashContext is invalid after calling this function. The length of the result buffer is fixed for each algorithm.

SV_HASH_MD5: 16 bytes.

SV_RANDOM

```
typedef struct tagSV_RANDOM
```

Random number generation service.

Generate()

```
bool_type (PRV_API* Generate)(unsigned char* data,  
    unsigned data_len);
```

Parameters Defined

Parameter	In / Out	Description
data	Out	Data to process, output buffer (must be aligned on 8-byte boundary)
data_len	In	Length of data required

Description

Generates random bits.

SV_GUID

```
typedef struct tagSV_GUID  
GUID generator service
```

CreateGuid()

```
bool_type (PRV_API* CreateGuid)(PRV_CHAR* const guid,  
    unsigned* const guid_len);
```

Parameters Defined

Parameter	In / Out	Description
guid	Out	Buffer to hold result guid
guid_len	In, Out	In: Size of guid. Out: Required size.

Description

Return a new GUID in the guid parameter.

GetPrivilegeInstallationGuid()

```
bool_type (PRV_API* GetPrivilegeInstallationGuid)(PRV_CHAR* const guid,  
    unsigned* const guid_len);
```

Parameters Defined

Parameter	In / Out	Description
guid	Out	Buffer to hold result guid
guid_len	In, Out	In: Size of guid. Out: Required size.

Description

Return the HASP SL Installation GUID. The HASP SL Installation is a GUID that is generated at the first installation of the HASP SL Runtime.

SV_ERRORCONTEXTKEEPER

```
typedef struct tagSV_ERRORCONTEXTKEEPER
```

This service is provided to keep a per-thread error context value.

Open()

```
bool_type (PRV_API* Open)(ErrorContextKeeperContext* ctx);
```

Parameters Defined

Parameter	In / Out	Description
ctx	Out	Context to use for other service functions

Description

Create a per-thread error context value.

Set()

```
bool_type (PRV_API* Set)(ErrorContextKeeperContext ctx,
    const err_ctx* e);
```

Parameters Defined

Parameter	In / Out	Description
ctx	In	Error context keeper instance to use
e	In	Error context. A value of 0 will clear the context.

Description

Sets the last error context for the calling thread.

Get()

```
bool_type (PRV_API* Get)(ErrorContextKeeperContext ctx);
```

Parameters Defined

Parameter	In / Out	Description
ctx	In	Error context keeper instance to use

Description

Returns the last error context for the calling thread. If a context has not been set, 0 will be returned.

Close()

```
const err_ctx* (PRV_API* Close)(ErrorContextKeeperContext ctx);
```

Parameters Defined

Parameter	In / Out	Description
ctx	In	Error context keeper instance to use

Description

Close a error context keeper. The error context is invalid after calling this function.

SV_MPI

```
typedef struct tagSV_MPI
```

Use the Memory Protection Interface (MPI) Service in your License Handler to protect your injected product files against dynamic attacks. When the HASP SL licensing data is injected into your protected file, it is fingerprinted, and the fingerprint is stored in a scan table.

At runtime, when your protected file is run, and scanning is turned on, your License Handler can scan the protected file, fingerprint the injected code, then compares it to the fingerprint values in the Scan table, which were created at the time of injection. If fingerprint matches, then file is allowed to run. If the scan fails, then the system terminates and the panic DLL loads. The Execution Environment also includes a scan table, thus enabling you to run Scan() against the HASP SL Execution Environment.

The reference implementations provide examples of how Scan() is used by the to prevent dynamic attacks. Scan() is most effective if called just before evaluating a cryptographic function.

Default MPI settings in Usage Builder workspace file (*.PRV)

You can enable or disable MPI for your products by setting parameters directly in your workspace file for each protected file. If you change the settings, then you must rebuild the HASP SL Setup File for your product in order for your changes to be built into your product files at activation.

Function	Value
MPIEnabled	True. If you set this to false, then no scanning happens, just as if you never wrote it into your code.
MPIScanPeriod	0. Enter a value from 0 - 100, where 0 = No auto scanning, and 100 = as much auto-scanning as possible.
MPITableRequired	False. This is set to false for backward compatibility. Previous versions of HASP SL do not have scanning tables. We recommend that you make this true in order to enforce scanning.
BadPlatformExecuteWithoutPLF	False Only applies to DLLs using the Win32 PE executable injection type of injection. See Injection .

Scan()

```
bool_type (PRV_API* Scan)();
```

Description

Scan the MPI tables of the protected file. If the EE was loaded not through a mainhook, scanning will do nothing.



If one of the files in the product MPI is protected then all files in the product are MPI protected. Hence if you need to remove MPI protection then you need to remove it from all files in the product.

SV_SYSTEMLOCK

```
typedef struct tagSV_SYSTEMLOCK
```

This is a recursive, system-wide, named mutual exclusion lock service. The Product Activation License Handler uses this service when using [Secure Storage Services Overview](#) to synchronize multiple instances of the same Protected File. This ensures that License Handler UIs will not display for multiple instances of the same Protected File.

Note that locks must be released.

Open()

```
bool_type (PRV_API* Open)(const char* name,
    SystemLockContext* ctx);
```

Parameters Defined

Parameter	In / Out	Description
name	In	Lock name
ctx	Out	Context of lock. This will be used for other service functions.

Description

Open a system-wide lock.

Lock()

```
bool_type (PRV_API* Lock)(SystemLockContext ctx);
```

Parameters Defined

Parameter	In / Out	Description
ctx	In	Context of lock.

Description

Grab the lock or block until it is released.

Unlock()

```
bool_type (PRV_API* Unlock)(SystemLockContext ctx);
```

Parameters Defined

Parameter	In / Out	Description
ctx	In	Context of lock.

Description

Release a previously grabbed lock.

Close()

```
bool_type (PRV_API* Close)(SystemLockContext ctx);
```

Parameters Defined

Parameter	In / Out	Description
ctx	In	Context of lock.

Description

Close a lock.

SV_URS

```
typedef struct tagSV_URS
```

This service enables you to load and unload runtime interfaces, such as the License Handler UI, from your own License Handler.

LoadPublisherURSInterface()

```
bool_type (PRV_API* LoadPublisherURSInterface) (LicenseCallbackContext
lcb_context,
    const PRV_CHAR* dll_name,
    const PRV_CHAR* itf_name,
    void** itf,
    DllHandle* dll_handle);
```

Parameters Defined

Parameter	In / Out	Description
lcb_context	In	Callback value for the Execution Environment context. The Execution Environment provides this value when your License Handler is started. See Start() . This value is used to call back to the Execution Environment.
dll_name	In	DLL name. For example, you would specify the name of your License Handler UI, which was provided by the Execution Environment in the license_gui_dll_name parameter of Start() .
itf_name	In	Name of the interface supported by the DLL being loaded. For example, you would specify the name of the interface supported by your License Handler UI.
itf	Out	Pointer to resulting interface (no memory is allocated here)
dll_handle	Out	Handle of loaded DLL.

Description

Load a publisher interface from a DLL during runtime. The DLL will be verified using the publisher's public key.

UnloadPublisherURSInterface()

```
bool_type (PRV_API* UnloadPublisherURSInterface) (LicenseCallbackContext
lcb_context,
    DllHandle dll_handle);
```

Parameters Defined

Parameter	In / Out	Description
lcb_context	In	Callback value for the Execution Environment context. The Execution Environment provides this value when your License Handler is started. See Start() . This value is used to call back to the Execution Environment.
dll_handle	In	Handle of loaded DLL

Description

Unload a publisher DLL.

SV_EC_INFO2

```
typedef struct tagSV_EC_INFO2
```

This service enables you to obtain information about a protected file.

GetECPath()

```
bool_type (PRV_API* GetECPath) (LicenseCallbackContext lcb_context,
    PRV_CHAR* ec_path,
    unsigned* ec_path_len);
```

Parameters Defined

Parameter	In / Out	Description
lcb_context	In	Callback value for the Execution Environment context. The Execution Environment provides this value when your License Handler is started. See Start() . This value is used to call back to the Execution Environment.
ec_path	Out	Path of the protected file.
ec_path_len	In / Out	In: ec_path buffer length. Out: required buffer size.

Description

Get the path of the protected file.

GetECCommandLine()

```
bool_type (PRV_API* GetECCommandLine) (LicenseCallbackContext lcb_context,
    PRV_CHAR* ec_cmd,
    unsigned* ec_cmd_len)
```

Parameters Defined

Parameter	In / Out	Description
lcb_context	In	Callback value for the Execution Environment context. The Execution Environment provides this value when your License Handler is started. See Start() . This value is used to call back to the Execution Environment.
ec_cmd	Out	Full command line.
ec_cmd_len	In / Out	In: ec_cmd buffer length. Out: required buffer size.

Description

Get the protected file's command line of the protected file's enclosing process.

GetExpressionElement()

```
bool_type (PRV_API *GetExpressionElement)(LicenseCallbackContext lcb_context,
    EE_LICEXPR_ELEM_INFO* const expr);
```

Parameters Defined

Parameter	In / Out	Description
lcb_context	In	Callback value for the Execution Environment context. The Execution Environment provides this value when your License Handler is started. See Start() . This value is used to call back to the Execution Environment.
expr	Out	Information about a License Expression Element See EE_LICEXPR_ELEM_INFO .

Description

Obtain the license expression element for the calling license handler

GetECInfo()

```
bool_type (PRV_API *GetECInfo)(LicenseCallbackContext lcb_context,
    EE_EC_INFO* const ec_info);
```

Parameters Defined

Parameter	In / Out	Description
lcb_context	In	Callback value for the Execution Environment context. The Execution Environment provides this value when your License Handler is started. See Start() . This value is used to call back to the Execution Environment.
ec_info	Out	Information about the protected file. See EE_EC_INFO .

Description

Obtain the licensing information about an EC

GetExpressionElementsForEC()

```
bool_type (PRV_API* GetExpressionElementsForEC)(const EE_EC_INFO* const ec,
    EE_LICEXPR_ELEM_INFO* const expr,
    unsigned* const expr_len);
```

Parameters Defined

Parameter	In / Out	Description
ec	In	Protected file to query. See EE_EC_INFO .
expr	Out	Information about a License Expression Element. See EE_LICEXPR_ELEM_INFO .
expr_len	In / Out	In: ec_cmd buffer length. Out: expected buffer size.

Description

Return all license expression elements for a given protected file.

SV_LOGGER

```
typedef struct tagSV_LOGGER
```

HASP SL Client Log Service. Log messages are written to the Kernel debugger console on Win32 and to the system log on Darwin

SV_LOGGER provide the following functions:

Function	Purpose
Log	Log a message to the HASP SL Runtime Log.

Log

```
void (PRV_API* Log)(const PRV_CHAR* message);
```

Parameter Defined

Parameter	In / Out	Description
message	In	Message to log.

Description

Log a message to the HASP SL Runtime Log.

Example Interface between Handler and Handler UI

This section documents the interface between Product Activation License Handler (prv_asm_lic.dll) and Product Activation Sample UI (prv_asm_lic_gui.dll) to provide a specific example of how a Handler UI works.

Product Activation Sample UI Interface

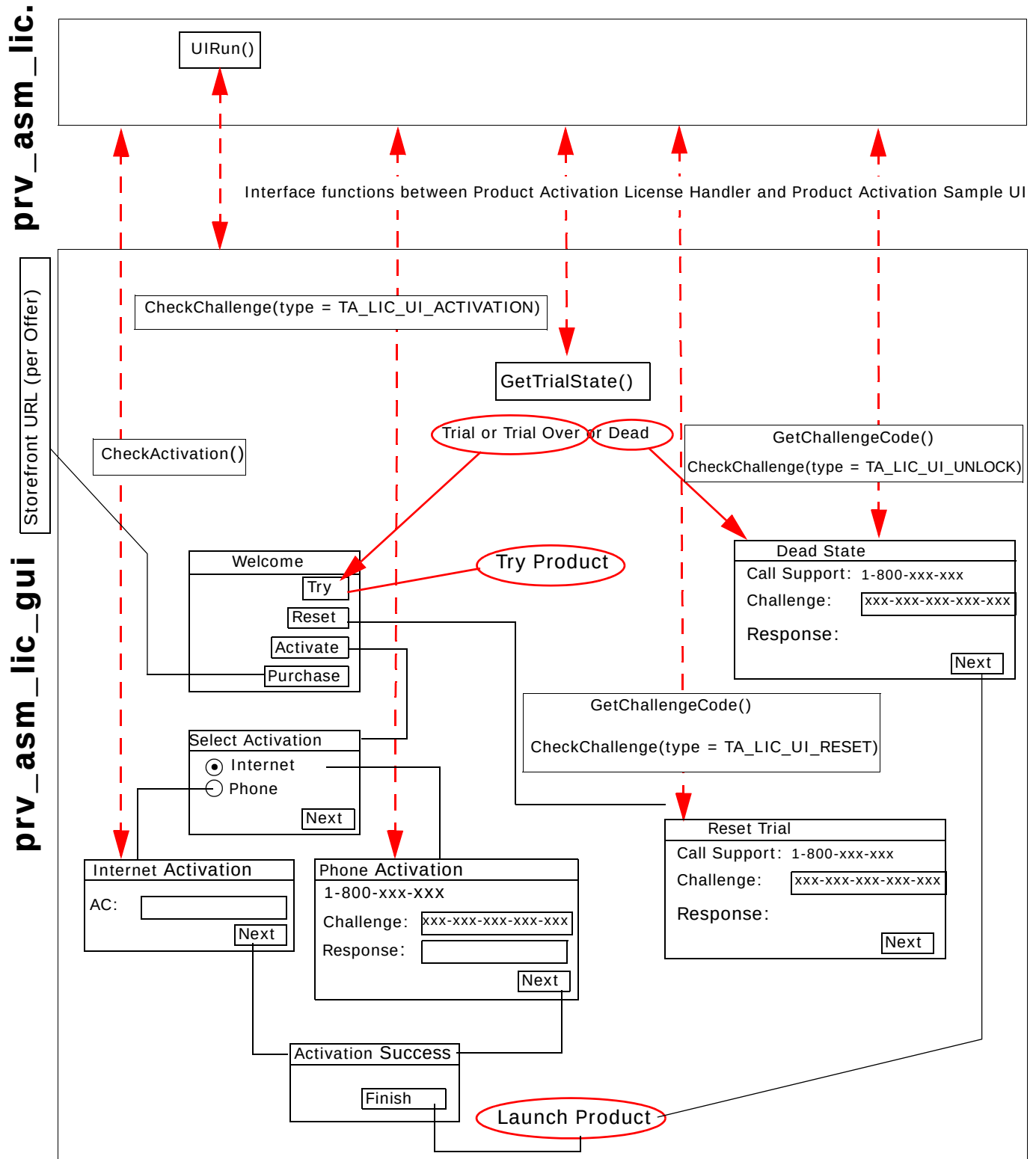
Figure 9 illustrates the function calls used by the Product Activation Sample UI to get the necessary information from the Product Activation License Handler to display to the end user. The following two interfaces are defined to communicate between these two components:

Function	Purpose
IASMLICENSEGUI	Must be supported by the Product Activation Sample UI in order to be run by the Product Activation License Handler.
IASMLICENSEGUI_CALLBACKS	This is the interface supported by the Product Activation License Handler in order to support the Sample UI calls for data.

Product Activation License Handler uses the [LoadPublisherURSInterface\(\)](#) function from the [SV_URS](#) service to load the License Handler UI.

Product Activation License Handler then essentially calls [UIRun\(\)](#) and the Product Activation Sample UI takes over from there.

Figure 9. Product Activation Sample UI Interfaces



IASMLICENSEGUI

```
typedef struct tagIASMLICENSEGUI
```

Interface of Product Activation Sample UI (prv_asm_lic_gui).

Initialize()

```
void (PRV_API *Initialize)(const IASMLICENSEGUI_CALLBACKS* pCallbacks,
    const IEE_SERVICES* services);
```

Parameters defined

Parameter	In / Out	Description
pCallbacks	In	Callbacks into license DLL.
services	In	EE service interface.

Description

Initialize the UI DLL. Any global data in the UI DLL should be initialized here. This function must not block.

UIRun()

```
bool_type (PRV_API *UIRun)(LicenseContext lic_context,
    bool_type more_licenses,
    UserInteractionResult* result);
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to use when calling the ILICENSEGUI_CALLBACKS
more_licenses	In	TRUE if there are more licenses in the license expression
result	Out	Result of interaction, which is returned when user dismisses GUI. The result will be one of the UserInteractionResult. This function should block until result returned.

Description

Start the UI message loop.

Finalize()

```
void (PRV_API *Finalize)();
```

Description

Unload the UI. This function must not block. This function is called in the same thread that called Initialize().

IASMLICENSEGUI_EX

```
typedef struct tagIASMLICENSEGUI_EX
```

Interface of Product Activation Sample UI (prv_asm_lic_gui).

SetLicenseCallbackContext()

```
void (PRV_API *SetLicenseCallbackContext)  
    (LicenseCallbackContext *lcb_context);
```

Parameters Defined

Parameter	In / Out	Description
lcb_context	In	License callback context to set.

Description

Sets the license callback context.

IASMLICENSEGUI2

```
typedef struct tagIASMLICENSEGUI2
```

Interface of Product Activation Sample UI (prv_asm_lic_gui).

SetCallbacks2()

```
void (PRV_API *SetCallbacks2)(const IASMLICENSEGUI_CALLBACKS2* pCallbacks2);
```

Parameters Defined

Parameter	In / Out	Description
pCallbacks2	Out	Initiates an IASMLICENSEGUI_CALLBACKS2 structure

Description

Sets the callback functions.

IASMLICENSEGUI3

```
typedef struct tagIASMLICENSEGUI3
```

Extended interface of the ASM License GUI DLL.

```
void (PRV_API *SetCallbacks3)(const IASMLICENSEGUI_CALLBACKS3* pCallbacks3);
```

Parameters Defined

Parameter	In / Out	Description
pCallbacks3	Out	Initiates an IASMLICENSEGUI_CALLBACKS3 structure

Description

Sets the callback functions.

IASMLICENSEGUI_CALLBACKS

```
typedef struct tagIASMLICENSEGUI_CALLBACKS
```

This is the interface to support calls from Product Activation Sample UI to Product Activation License Handler.

In Aladdin's Product Activation License model, [IASMLICENSEGUI_CALLBACKS](#) defines the interfaces that the `prv_asm_lic_gui.dll` (Product Activation Sample UI) calls to the `prv_asm_lic.dll` (Product Activation License Handler) to get the information it requires to display the correct information to the end user. This interface is defined in the `asm_license_api.h`, which is shared by both License Handler and License Handler UI.

If you want to change the flow of the Product Activation Sample UI, then you will need to modify the code in `prv_asm_lic_gui.dll` to redefine when or which interfaces are called back to `prv_asm_lic.dll`.



The [IASMLICENSEGUI_CALLBACKS](#) interface is not defined by the HASP SL Framework, but rather an arbitrary interface between the license handler and the license UI. The HASP SL Framework has no idea what a License, License Handler, or License UI is. All Licenses and License Handlers, License UIs are publisher-defined entities, which are transparent to the HASP SL Framework. You may modify the Product Activation model we've provided in the Visual C++ project in the SDK to fit your software activation needs, or define your own License Handler to License UI

GetTrialState()

```
bool_type (PRV_API* GetTrialState)(LicenseContext lic_context,  
    ASM_LICENSE_STATUS* current_status);
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance
current_status	Out	Trial state, one of ASM_LICENSE_STATE .

Description

Obtain the current trial state data. The current trial state data informs the GUI what to display to the user, such as whether or not the Trial button is grayed out. Trial data also includes license parameters, such as whether or not to display the Buy button. See [Product Activation Sample UI Interfaces](#).

ASM_LICENSE_STATUS

```
typedef struct tagASM_LICENSE_STATUS
```

License status data returned to the License Handler UI via the [GetTrialState\(\)](#) callback to the Product Activation License Handler.

Value	Description
unsigned csize	Size of structure.
ASM_LICENSE license	The actual license.
ASM_TRIAL_CONSUMPTION trial_consumption	Current trial consumption. See ASM_TRIAL_CONSUMPTION .
ASM_ACTIVATION_DATA activation_data	License activation data.
bool_type consumption_valid	Boolean to signal whether the consumption is valid.
bool_type clock_tampered	Boolean to signal if the system clock is in the "tampered" state.
ASM_LICENSE_STATE state	See ASM_LICENSE_STATE .

ASM_LICENSE

```
typedef struct tagASM_LICENSE
```

Product Activation trial data passed in the [ASM_LICENSE_STATUS](#) structure.

Value	Description
csize	Size of structure.
ASM_TRIAL_OPTIONS	Bit mask specifying which trial options are used: 1.Fixed number of days ASM_LIC_DAYS = 1 2.Fixed date to run until ASM_LIC_EXPIRATION = 2 3.Fixed number of executions ASM_LIC_EXECUTIONS = 4 4.Fixed number of minutes to execute ASM_LIC_EXEC_TIME = 8
days	Days to use.
execs	Number of executions.
usage	Number of minutes to run.
PRV_DATE expiry	Expiration date (local time).
cshift	The maximal amount of clock shift accepted by the license as valid.
trial_activation_url	Trial activation server (255 ANSI characters)

Value	Description
trial_registration_url	Trial activation server (255 ANSI characters).
registration_optional	Whether End-User can cancel out of trial.
perp_activation_url	Perpetual Activation server URL (255 ANSI characters).
purchase_url	Purchase URL (255 ANSI characters).
admin_server	Administration server.

ASM_TRIAL_CONSUMPTION

```
typedef struct tagASM_TRIAL_CONSUMPTION
```

Product Activation trial consumption status.

Value	Description
unsigned cbsize	Size of structure.
unsigned execs	Number of executions already executed.
unsigned usage	Number of minutes already ran.
PRV_DATE invocation_date	Date trial was invoked (UTC).
PRV_DATE last_known_date	Latest known time (UTC). It must be true that (<i>current time</i>) - (<i>max_clock_can_go_back</i>) > (<i>last_known_date</i>) or license fails.
bool_type trial_registered	Pre trial state: Trial already registered.
bool_type trial_activated	Pre trial state: Trial already activated.

ASM_LICENSE_STATE

```
typedef enum tagASM_LICENSE_STATE
```

Possible Product Activation License states that the License Handler will return to the License Handler UI.

Value	Description
ASM_LIC_STATE_NONE	No state.

Value	Description
ASM_LIC_STATE_PRE_TRIAL	An action is required before the trial, such as a trial activation, a trial registration or both.
ASM_LIC_STATE_TRIAL	In trial.
ASM_LIC_STATE_TRIAL_OVER	Trial over: limits exceeded or corrupt storage.
ASM_LIC_STATE_ACTIVATED	Activated.
ASM_LIC_STATE_DEAD	Dead state. In Aladdin's Product Activation implementation, the dead state is achieved only after a product has been activated, and a tampering is detected.



Note that there are two other states stored: ASM_LIC_STATE_NONE, and ASM_LIC_STATE_ACTIVATED. However, these two states will never be provided to the GUI.

ASM_LIC_STATE_NONE is returned by the Execution Environment to the License Handler the first time a Protected File is run. The License Handler immediately sets the state to Trial prior to initializing the GUI.

ASM_LIC_STATE_ACTIVATED indicates that a License-Product has been activated. By design, the GUI will not display for activated products.

CheckPerpetualActivation()

```
bool_type (PRV_API* CheckPerpetualActivation)(LicenseContext lic_context,
    const char* activationCode,
    const char* distributionCode);
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance
activationCode	In	Activation code (up to MAX_ASM_LEN bytes length)
distributionCode	In	Distribution code (up to MAX_ASM_LEN bytes length)

Description

Check a perpetual activation code. If the check is successful, the license state is changed to "activated."

CheckTrialActivation()

```
bool_type (PRV_API* CheckTrialActivation)(LicenseContext lic_context,
    const char* activationCode,
    const char* distributionCode);
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance
activationCode	In	Activation code (up to MAX_ASM_LEN bytes length)
distributionCode	In	Distribution code (up to MAX_ASM_LEN bytes length)

Description

Check a trial activation code. If the check is successful, the license state is changed to "activated."

RegisterTrial()

```
bool_type (PRV_API* RegisterTrial)(LicenseContext lic_context,
    const char* campaignCode,
    bool_type do_registration)
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance
campaignCode	In	Campaign code to use
do_registration	In	Send registration

Description

Register a trial.

GetPhoneChallengeCode()

```
bool_type (PRV_API* GetPhoneChallengeCode)(LicenseContext lic_context,
    ASM_LIC_PHONE_CHALLENGE_TYPE type,
    unsigned char* challengeCode)
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance
type	In	Type of challenge to obtain
challenge	Out	Challenge code (8 bytes length)

Description

Obtain a phone challenge code.

CheckPhoneChallenge()

```
bool_type (PRV_API* CheckPhoneChallenge)(LicenseContext lic_context,  
    ASM_LIC_PHONE_CHALLENGE_TYPE type,  
    const unsigned char* challengeCode,  
    const unsigned char* responseCode)
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance
type	In	Type of challenge to check
challengeCode	In	Challenge (8 bytes length)
responseCode	In	Response (8 bytes length)

Description

Check a phone challenge code. If the check is successful, the license state is changed to "Activated," or "Trial" depending on type of challenge code being checked. See [License State](#).

GetLastError()

```
const err_ctx* (PRV_API* GetLastError)();
```

Description

Return the context for the last error resulting from calling functions in this interface.

IASMLICENSEGUI_CALLBACKS2

```
typedef struct tagIASMLICENSEGUI_CALLBACKS2
```

This interface augments the functionality of [IASMLICENSEGUI_CALLBACKS](#).

UserRegistered()

```
bool_type (PRV_API* UserRegistered)(LicenseContext lic_context);
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance

Description

Checks if the user is registered.

RegisterUser()

```
bool_type (PRV_API* RegisterUser)(LicenseContext lic_context,
    const char* firstName,
    const char* lastName,
    const char*emailAddress);
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance
firstName	In	
lastName	In	
emailAddress	In	

Description

Sends user registration data to the server.

AddRegDataW

(win32 only)

```
bool_type (PRV_API* AddRegDataW)( const U_CHAR* wstrKey,
    const U_CHAR* wstrVal,
    RegContextHandler hRegCtx);
```

Parameters Defined

Parameter	In / Out	Description
wstrKey	In	Registration data key
wstrVal	In	Registration data value
hRegCtx	In	Handler to registration context

Description

Accumulates registration data - the Unicode version.

CommitRegistration

```
bool_type (PRV_API* CommitRegistration)(const SA_OFFER_INFO* offer_info,  
                                         RegContextHandler hRegCtx);
```

Parameters Defined

Parameter	In / Out	Description
offer_info	In	Offer for which to record data.
hRegCtx	In	Handler to registration context.

Description

Finalizes the add_reg_data sequence and commits on the server.

IASMLICENSEGUI_CALLBACKS3

```
typedef struct tagIASMLICENSEGUI_CALLBACKS3
```

This interface augments the functionality of [IASMLICENSEGUI_CALLBACKS](#).

UserRegisteredLocally

```
bool_type (PRV_API* UserRegisteredLocally)(LicenseContext lic_context);
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance

Description

Checks if the user is registered.

CheckPhoneChallengeWithLicData

```
bool_type (PRV_API* CheckPhoneChallengeWithLicData)( LicenseContext
lic_context,
    const unsigned char* challengeCode,
    const unsigned char* responseCode,
    const unsigned char* licenseData);
```

Parameters Defined

Parameter	In / Out	Description
lic_context	In	Context to identify license instance
challengeCode	In	Challenge
responseCode	In	Response
licenseData	In	License data

Description

Check a Phone challenge. If the check is successful, the license state is changed Boolean to signal whether the consumption is valid accordingly.

Feature-based Licensing

This section uses an example to explain how feature-based licensing may be implemented using the ISDK Interface. See [ISDK Interface Reference](#).

Example Use Case: Eval and Full

Description of Licensing Requirements

Example Product Requirements

- Protection from illegal usage.
- Distribute reduced-functionality evaluation version and traditional boxed version (fully functional).

Example solution

Summary

Rather than create and distribute two versions of your code, which incurs cost in Development, QA, *and* Support, HASP SL enables you to create a single version of your code and distribute two licenses: one for Eval purposes and one for traditional sales-channel purposes:

License Name	Description
SP Eval	90-day limited trial, no activation, registration, or server notification necessary to begin trial.
SP Full	Activation required, 3-day grace (trial) after first execution.

The licenses govern the usage of your product. Read the following for additional comments concerning each requirement.

Protection from Illegal Usage.

HASP SL's injection technology secures your product, and activation prevents piracy.

Distribute Reduced-functionality Evaluation Version from Company Web Site.

Licensing your application as a whole entity does not require any code modification. However, licensing functionality within your application requires that your code be modified to call the ISDK interface provided by the HASP SL SDK library.

You have several options to license the evaluation version:

- Limited trial usage of evaluation? 90 days? 20 executions? et cetera
If defining a trial period, you then have an additional option:
 - Require activation prior to beginning trial period
 - Notify server that trial has started.
 - Provide Buy button, enabling customers to purchase Full license.

- Unlimited usage of evaluation.

You may feel that limiting feature usage is sufficient without limiting a trial period.

Distribute Traditional Boxed Version (fully functional).

The only thing you need to add to your boxed product is a valid activation code. The HASP SL Administration Server includes an activation code generator. Users must enter a valid activation code to 'unlock' their product.

How to Set Up your Product with an Eval and Full License

Step 1: Create HASP SL Definition of your Product

	Item	Example values	Description
Prepare your product	Product name	Super Product	
	Modify application code	superProduct.exe	Link to HASP SL SDK library, and use ISDK functions to check License ID prior to executing feature code. See Figure 10 . See also ISDK Interface Reference .
Set up server	Define Offer data	Offer Name = Super Product Full Offer Usage = perpetual	Offers must be created for licenses requiring activation. See the <i>Commerce Server Administrator User Guide</i> for details Offer and activation code creation.
	Generate Activation Codes		Send the activation codes for your Product to your boxed goods fulfillment center. Each box needs a unique activation code.
Modify GUI	License Handler GUI	myCo_tryAndDie_gui.dll	Modify the text and images of the Try & Die License Handler GUI Reference Implementation to support the 'Evaluation of Super Product.'
		myCo_prodActivate_gui.dll	Modify the text and images of the Product Activation License Handler GUI Reference Implementation to support the 'Activation of Super Product.'

Define product in Usage Builder

	Item	Example values	Description									
	Inject files	superProduct.exe	An injected file is called a Protected File. Injecting the main executable prevents it from running without a valid license. Inject all of the files containing features to be licensed.									
	Create Licenses	name = "SP Eval"	Use Try & Die License Generator: 90-day limited trial. No activation, registration, or server notification is needed to begin trial.									
		name = "SP Full"	Use Product Activation License Generator with Offer "Super Product Full." Activation is required with a 3-day grace (trial) after first execution.									
	Define Product "Super Product Full"		See HASP <i>Usage Builder User Guide</i> for details.									
		Create License Expression for superProduct.exe: <table><tr><th>License</th><th>Product</th><th>Behavior</th></tr><tr><td>SP Full</td><td>Super Product Full</td><td>Active</td></tr><tr><td>SP Eval</td><td>Super Product Eval</td><td>Active</td></tr></table>	License	Product	Behavior	SP Full	Super Product Full	Active	SP Eval	Super Product Eval	Active	At runtime, there may exist one and only one License Expression for each Protected File. Thus, at build time, it's best to include all known unique License + Product combinations. This prevents future installs from breaking a binding to a previously activated license.
	License	Product	Behavior									
	SP Full	Super Product Full	Active									
	SP Eval	Super Product Eval	Active									
		Define License Handler binding = <table><tr><th>License + Product</th><th>License Handler</th><th>License Handler GUI</th></tr><tr><td>"SP Full" + "Super Product Full"</td><td>prv_asm_lic.dll</td><td>myCo_prodActivate_gui.dll</td></tr></table> prv_asm_lic.dll is the Product Activation License Handler that ships with the Usage Builder.		License + Product	License Handler	License Handler GUI	"SP Full" + "Super Product Full"	prv_asm_lic.dll	myCo_prodActivate_gui.dll			
	License + Product	License Handler	License Handler GUI									
	"SP Full" + "Super Product Full"	prv_asm_lic.dll	myCo_prodActivate_gui.dll									
	Add "SP Full" License, define runtime ID = "SP Full"											
Define Product "Super Product Eval"												
	Create License Expression for superProduct.exe: <table><tr><th>License</th><th>Product</th><th>Behavior</th></tr><tr><td>SP Full</td><td>Super Product Full</td><td>Active</td></tr><tr><td>SP Eval</td><td>Super Product Eval</td><td>Active</td></tr></table>	License	Product	Behavior	SP Full	Super Product Full	Active	SP Eval	Super Product Eval	Active		
License	Product	Behavior										
SP Full	Super Product Full	Active										
SP Eval	Super Product Eval	Active										
	Define License Handler binding = <table><tr><th>License + Product</th><th>License Handler</th><th>License Handler GUI</th></tr><tr><td>"SP Eval" + "Super Product Eval"</td><td>prv_td_lic.dll</td><td>myCo_tryAndDie_gui.dll</td></tr></table> prv_td_lic.dll is the Try & Die License Handler that ships with the Usage Builder.		License + Product	License Handler	License Handler GUI	"SP Eval" + "Super Product Eval"	prv_td_lic.dll	myCo_tryAndDie_gui.dll				
License + Product	License Handler	License Handler GUI										
"SP Eval" + "Super Product Eval"	prv_td_lic.dll	myCo_tryAndDie_gui.dll										
	Add "SP Eval" License, define runtime ID = "SP Eval"											

Step 2: Build HASP SL Install Files for Defined Products

For each of product, click the Build button to generate the HASP SL Setup File.

- Super Product Full: Super Product Full_prv_setup.exe
- Super Product Eval: Super Product Eval_prv_setup.exe

Step 3: Update your Product's Original Installer and Distribute your Product

Replace the files in your product with the injected files, then modify your product installer to run the HASP SL Setup File. In this scenario, you will probably create two installers: one for Eval and one for Full.

Distribution is very flexible. You can ship a CD of each license-managed product: Eval CD and Full CD.

Upgrading from Eval to Full

To transition your customers from the evaluation usage of your product to the fully functioning product, you need only ship them the HASP SL Installation file "Super Product Full_prv_setup.exe," with a valid activation code. You do not need to re-distribute the product itself.

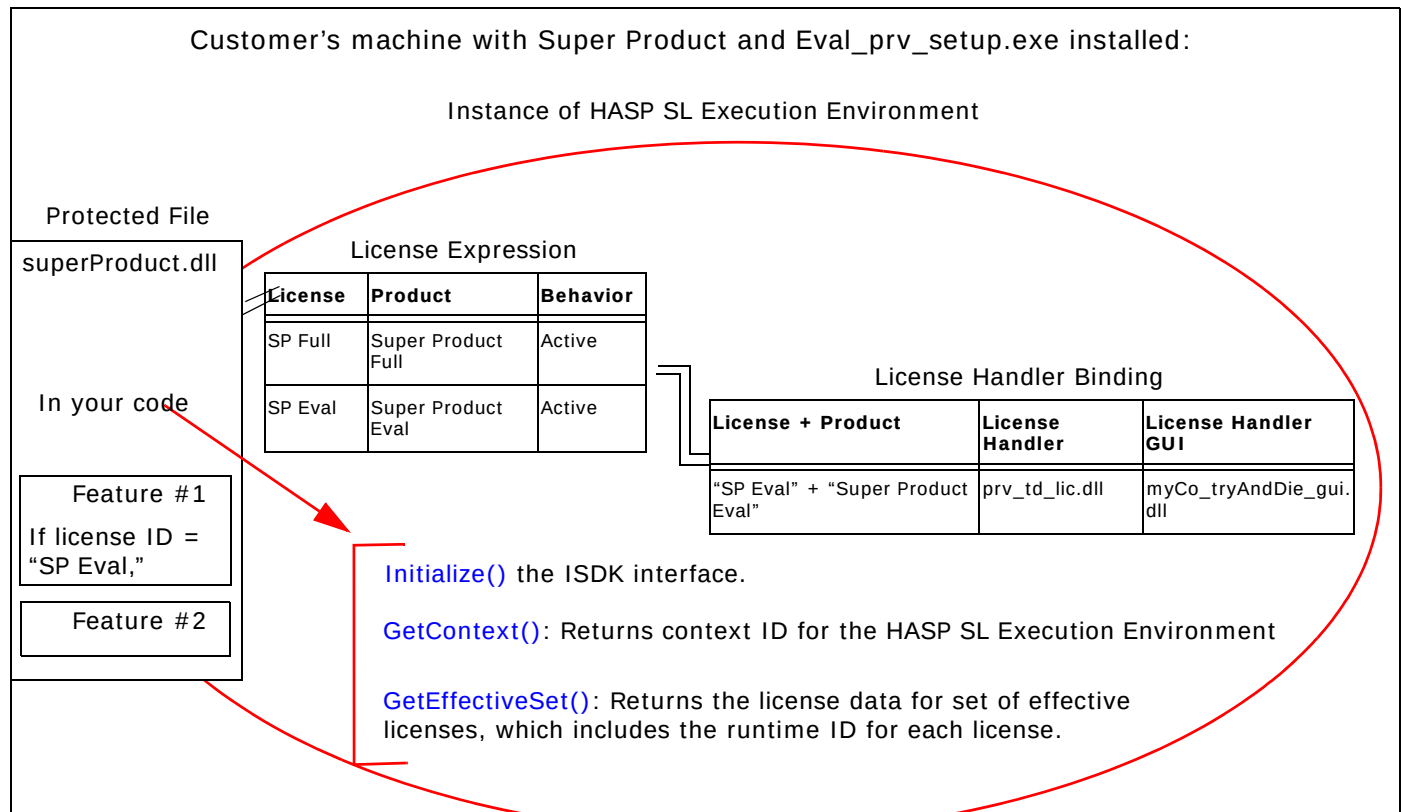
ESD distribution

Any tree directory of files on a CD may be distributed via ESD. HASP SL provides a tool called the Delivery Manager, which enables the "same as CD" distribution of files via the internet. "Same as CD" means that the distribution of the files are not wrapped, or packaged in any way, but simply delivered, then checksum verified. Your customers may burn them to a CD after download to have a CD with the exact contents of the CD shipped in the box. Because your product is secure, it is not necessary to secure your distribution. See the *HASP Delivery Manager for Windows Guide* or *HASP Delivery Manager for Macintosh Guide* for setup instructions.

Runtime Description of the Eval and Full License

Understanding the HASP SL Licensing Framework runtime logic will help you setup your product definitions. The steps below describe the HASP SL Licensing Framework at runtime.

Figure 10. Eval License at runtime



1. The end user runs your Super Product application.
2. The injected superProduct.exe runs the HASP SL [Execution Environment](#).
3. The Execution Environment locates the [License Expression](#) defined for superProduct.exe, which is in the [ProtectedFile2License Binding](#).
4. The Execution Environment then processes the License Expression to find the effective set of licenses.

In our runtime example, the Execution Environment determines that the license "SP Full" is not installed. It then verifies and starts license "SP Eval" to determine that it is effective, thus "SP Eval" shall govern Super Product for this invocation. See also [How does the Execution Environment identify the effective License set?](#).

5. From the data in the License Expression, the Execution Environment locates, verifies, and starts License Handler "prv_td_lic.dll" which was defined to handle license "SP Eval" for product "Super Product Eval."
6. prv_td_lic.dll interprets the terms defined in license "SP Eval," runs myCo_tryAndDie_gui.dll, which checks the state of the trial, then displays the appropriate screen to the user, such as a 'Welcome' message and current status (for example, Day 3 of 90-day). Using Aladdin's Try & Die License Model, the user must click a Try button to continue the evaluation. See also [Example Interface between Handler and Handler UI](#).

When the trial period expires, prv_td_lic.dll refuses to allow superProduct.exe to run.

7. After superProduct is “OK'd” by the License Handler to run, it may communicate directly with the HASP SL Execution Environment via the ISDK interface to get the effective license set ([Eval License at runtime](#)).

Your code can now allow or not allow the use of your features based on the installed license. When both licenses reside on your customers machine, the HASP SL Execution Environment will always try to use license “SP Full” first because of the defined order License Expression elements for superProduct.exe.

SDK Interface Loader

The SDK Interface Loader supplies the functionality to manage the loading and unloading of the various SDK interfaces. Loading and Unloading of the various interfaces is used for [Feature-based Licensing](#).

List of SDK Interface Loader Functions

The following functions are available for use with HASP SL SDK Library.

Name	Description
PRV_SDK_LoadExecutionEnvironment	Load the execution environment explicitly into the process address space.
PRV_SDK_UnloadExecutionEnvironment	Unload the execution environment from the process address space.
PRV_SDK_LoadSDKInterface	Load the SDK interface for an EC. See ISDK Interface Reference .
PRV_SDK_UnloadSDKInterface	Unload the SDK interface. See ISDK Interface Reference .
PRV_SDK_LoadServiceInterface	Load the SDK interface for this EC. See Execution Environment: Interface and Services .
PRV_SDK_UnloadServiceInterface	Unload the ISERVICES interface. See Execution Environment: Interface and Services .
PRV_SDK_FixServiceInterface	Fix an interface obtained from IEE_SERVICES::GetServiceInterface() to match the standards of Macintosh machines. A fixed interface must be unfixed via PRV_SDK_UnfixServiceInterface() .
PRV_SDK_UnfixServiceInterface	Unfixes an interface fixed by PRV_SDK_FixServiceInterface() .
PRV_SDK_GetLastError	Returns the last error for the calling thread for the SDK library.

ISDK Interface Reference

Overview

The purpose of the ISDK interface is to enable publishers to implement feature-based licensing (see [License Data Implementation](#)), or otherwise control when licenses are verified. For example, publishers may use the ISDK interface to stop usage of an application after 30 minutes, while the application is still running.

The ISDK interface is provided in the HASP SL SDK library (Windows: `prv_sdk.lib`; Macintosh: `prv_sdklib`), which is installed in the following location (by default): `C:/Program Files/Aladdin/HASP SL/Client SDK/LIB`.



The ISDK interface makes use of the PSAPI.dll. By default, this DLL does not exist in Windows NT4 SP6, which is supported by HASP SL. If your protected application is going to run on Windows NT 4 SP6, you must redistribute this DLL, which is available for download and redistribution from the Microsoft web site at <http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=3D1FBAED-D122-45CF-9D46-1CAE384097AC>

The ISDK interface provides your program access to the HASP SL Execution Environment, which can provide to your application the following types of HASP SL-managed runtime data:

- [License Expression](#)
- [License](#)
- [License Handler](#)
- [License Handler UI](#)

Note the licensing potential of function [TransmitToLicenseDLL\(\)](#), which enables your application to communicate directly with your License Handler at runtime.

This can provide a powerful and dynamic feature-base licensing solution whereby your application can query the licensing system, checking installed license files to determine which features may be used or not, checking trial state, saving data to secure storage, etc. ISDK provides complete control over when, what, and how of licensing for a publisher's product.



The ISDK interface itself does not provide any security, thus your application files containing ISDK calls need to be securely injected.

How does it work?

First, in the code of the files you want to license, you must statically link to the provided stub library, `prv_sdk.lib` (Win), or `prv_sdklib` (Mac). Then use the ISDK functions to query the licensing system (HASP SL Execution Environment) as appropriate to your licensing needs. You must evaluate the effective set of licenses in order to ensure that your product is protected.



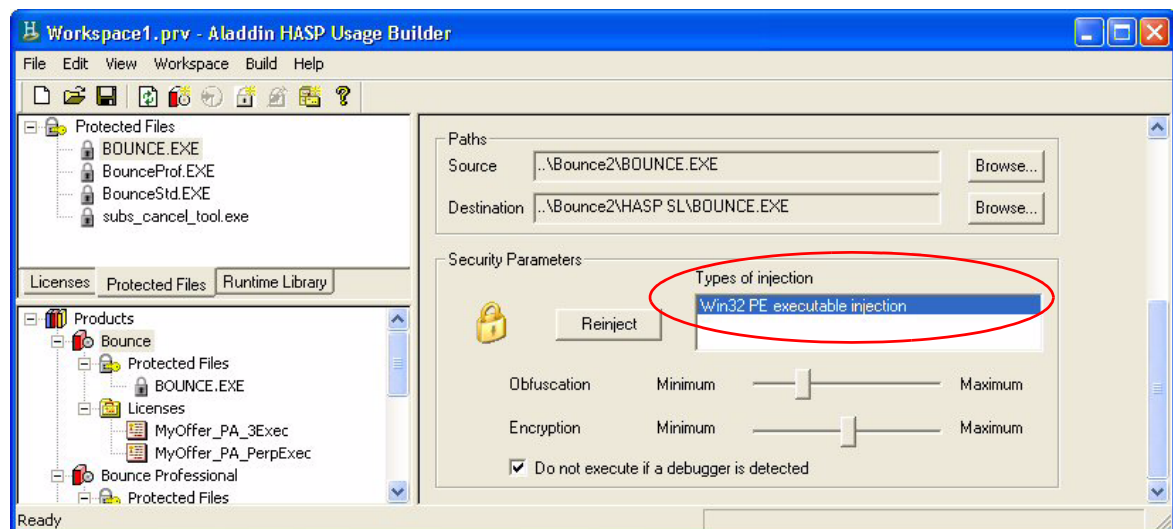
When using the ISDK functions in DLL files, do not call functions from DLL main. This is because you may not load a DLL from DLL main, and the HASP SL Execution Environment is a DLL.

Injection

After linking to the HASP SL SDK Library and using the ISDK functions in your code, you must add your files to your Usage Builder workspace, and then inject them. You build your licenses and product as you would if you were not using the SDK. See the *Usage Builder User Guide* for details.

At build time, when you inject, your application files are scanned and the library stub is located. When the library is located, the Builder provides two additional injection choices.

Figure 11. Usage Builder: Types of injection



Injection Type Descriptions

Platform neutral SDK based (INSECURE)

This type of injection embeds the necessary licensing data into the stub library. You may inject Macintosh or Windows files. No security (encryption, obfuscation, or new main hook) is provided at all. The injected file is not modified at all. The purpose for this type of injection is for debugging your licensed applications.

When using this type of injection, you are responsible for using the ISDK functions to evaluate the effective set of licenses, and check license status.

Win 32 PE executable injection

This type of injection is the secure Windows injection that works on Windows files of type .exe, .dll, .ocx. This is the recommended type of injection whether or not you are using the SDK.

When using this type of injection, the effective set of licenses is evaluated by the injected code prior to allowing the file to run or DLL to load.

Win 32 PE explicit injection, SDK based (SECURE)

This type of injection provides the same security as *Win 32 PE executable injection*. However, the injected code will not evaluate the effective set of licenses, but rather your code is responsible for using the ISDK functions to evaluate the effective set of licenses.

The advantage of using this type of injection, is that you have control over when licenses are verified.

List of ISDK Functions

```
typedef struct tagISDK
```

The following functions are available for use with HASP SL SDK Library. The ISDK interface is a structure with a list of function pointers named ISDK.

ISDK function called	Description
Initialize()	Initializes your ISDK interface in the HASP SL Execution Environment.
GetContext()	Gets a context for secure communication with the HASP SL Execution Environment.
GetEffectiveSet()	Returns the effective license, which was used by the Execution Environment to allow this Protected File to run. See also How does the Execution Environment identify the effective License set? .
TransmitToLicenseDLL()	Transmit a message to your License Handler. To govern the print feature, for example, you would send a 'Can I print?' message to your License Handler.
GetECInfo()	Optional call to get runtime data about a Protected File, which is required to call GetExpressionForEC() .
GetExpressionForEC()	Optional call to get the License Expression and associated data defined for the Protected File.
GetLastError	Return last error context.
Finalize()	Finalizes the ISDK interface.



Macintosh

Concerning ISDK development: All of the header files assume that enums are 32-bit wide, which is true in VC++. However, when developing with Code Warrior 7.0, this is not true, and therefore you must set enums to 32-bits in your project settings.

ISDK functions

Initialize()

```
bool_type (PRV_API * Initialize)()
```

Description

Initializes the ISDK interface.

Parameters Defined

None.

Finalize()

```
void (PRV_API *Finalize)()
```

Description

Finalizes the ISDK interface.

Parameters Defined

None.

GetContext()

```
bool_type (PRV_API *GetContext)(const PRV_CHAR* ec_path,  
    SDKContext* sdk_context)
```

Description

Obtain an SDK context in order to securely communicate with the HASP SL Execution Environment.

Parameters Defined

Parameter	In / Out	Description
ec_path	In	Full path to the Protected File.
sdk_context	Out	Context placed here.

GetEffectiveSet()

```
bool_type (PRV_API *GetEffectiveSet)(SDKContext sdk_context,  
    EE_LICEXPR_ELEM_INFO* const expr,  
    unsigned* const expr_len)
```

Description

Obtain the effective license set (as a hash values) that will govern the protected file. The HASP SL Execution Environment identifies the effective license set by evaluating the license expression for the protected file. See [How does the Execution Environment identify the effective License set?](#).

Parameters defined

Parameter	In / Out	Description
sdk_context	In	Context to reference. See GetContext() .
expr	Out	Effective set of the pre-defined license expression. The values returned here are the hash values of licenses. See EE_LICEXPR_ELEM_INFO .
expr_len	In/Out	Size of expr buffer, Out: expected size.

TransmitToLicenseDLL()

```
bool_type (PRV_API *TransmitToLicenseDLL)(SDKContext sdk_context,
    const EE_LICEXPR_ELEM_INFO* lee,
    const unsigned char* in,
    unsigned in_length,
    unsigned char* out,
    unsigned* out_length);
```

Description

Transmit a message to your License Handler. HASP SL places no limits on this function. You may transmit any number of times, in any format or content you need. The dialog between your Protected File and your License Handler is entirely up to you.

The Transmit call is forwarded to your License Handler by the HASP SL Execution Environment via the Receive() function in the ILICENSE2 interface. You will need to write your License Handler such that it is designed to handle each of the messages coming from your Protected File.

Parameters Defined

Parameter	In / Out	Description
sdk_context	In	Context to reference. See GetContext() .
lee	In	Data returned by GetEffectiveSet() . See EE_LICEXPR_ELEM_INFO .
in	In	Data to send.
in_length	In	Length of data to send.
out	Out	Data to receive (allocated by caller).
out_length	In, Out	Length of data to receive.

GetECInfo()

```
bool_type (PRV_API *GetECInfo)(SDKContext sdk_context,
    EE_EC_INFO* const ec_info)
```

Description

Obtain HASP SL runtime information about the Protected File, which is necessary to call [GetExpressionForEC\(\)](#).

Parameters Defined

Parameter	In / Out	Description
sdk_context	In	Context to reference. See GetContext() .
ec_info	Out	Protected File information here. See EE_EC_INFO below.

EE_EC_INFO

```
typedef struct tagEE_EC_INFO{}
```

Information about a Protected File.

unsigned cbsize	Size of structure
PRV_CHAR name[EEINFO_MAX_STRING]	Name of Protected File. EEINFO_MAX_STRING is 255 characters.
PRV_CHAR hash[EEINFO_MAX_STRING]	Hash of Protected File. EEINFO_MAX_STRING is 255 characters.
EE_EC_STATUS status	Protected File status. See EE_EC_STATUS below.

EE_EC_STATUS

```
typedef enum tagEE_EC_STATUS{}
```

State of a Protected File.

EE_EC_STATUS_COMPLETE	Expression is complete
EE_EC_STATUS_MISSING_ECLICBINDING	ProtectedFile2License Binding is missing
EE_EC_STATUS_BROKEN_ECLICBINDING	ProtectedFile2License Binding is broken (tampered)

GetExpressionForEC()

```
bool_type (PRV_API* GetExpressionForEC)(const EE_EC_INFO* const ec,
    EE_LICEXPR_ELEM_INFO* const expr,
    unsigned* const expr_len)
```

Description

Returns the license expression for a given Protected File. Each Protected File may only have one license expression.

Parameters defined

Parameter	In / Out	Description
ec	In	Protected File to query. EE_EC_INFO is returned by GetECInfo() .
expr	Out	Put license expression here. See EE_LICEXPR_ELEM_INFO .
expr_len	In, Out	In: size of expr buffer Out: expected size

EE_LICEXPR_ELEM_INFO

```
typedef struct tagEE_LICEXPR_ELEM_INFO{}
```

Information about a License Expression.

unsigned cbsize	Size of structure
product	Product information. EE_PRODUCT_INFO
license	License information EE_LIC_INFO
behavior	Behavior. Behavior Modifier values used by the HASP SL Reference Implementations: 1 = Active, 0 = Passive. Your License Model may define many license behaviors.
status	Status. See EE_LICEXPR_STATUS .

EE_PRODUCT_INFO

```
typedef struct tagEE_PRODUCT_INFO{}
```

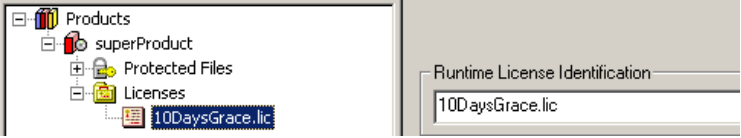
Information about a Product.

unsigned cbsize	Size of structure
PRV_CHAR name[EEINFO_MAX_STRING]	Name of Product. EEINFO_MAX_STRING is 255 characters.
PRV_CHAR sss[EEINFO_MAX_STRING]	Hash of Product. EEINFO_MAX_STRING is 255 characters.(sss = Secure Storage Selector)

EE_LIC_INFO

```
typedef struct tagEE_LIC_INFO{}
```

Information about a License.

unsigned cbsize	Size of structure
name[EEINFO_MAX_STRING]	Name of License. EEINFO_MAX_STRING is 255 characters. Note that this name is defined in the Usage Builder at the time that a license is added to a product. Select the license in the Product Pane (lower left) of the Usage Builder to view the Runtime identification value: 
hash[EEINFO_MAX_STRING]	Hash of License. EEINFO_MAX_STRING is 255 characters.
dll	License Handler DLL. See EE_URS_OBJECT_INFO .
ui_dll	License Handler UI DLL. See EE_URS_OBJECT_INFO .
publisher_urs	Publisher URS. See EE_URS_INFO .

EE_URS_OBJECT_INFO

```
typedef struct tagEE_URS_OBJECT_INFO{}
```

Information about a object.

unsigned cbsize	Size of structure
name[EEINFO_MAX_STRING]	Name of Object. EEINFO_MAX_STRING is 255 characters.
version	Version of object
display_info	Display information

EE_LICEXPR_STATUS

```
typedef enum tagEE_LICEXPR_STATUS{}
```

State of a license expression.

EE_LICEXPR_STATUS_COMPLETE	License Expression is complete.
EE_LICEXPR_STATUS_MISSING_LICENSE	License is missing.
EE_LICEXPR_STATUS_BROKEN_LICENSE	License is broken (tampered).
EE_LICEXPR_STATUS_MISSING_LICDLLBINDING	Product2LicenseFiles Binding is missing
EE_LICEXPR_STATUS_BROKEN_LICDLLBINDING	Product2LicenseFiles Binding is broken (tampered)

EE_LICEXPR_STATUS_MISSING_LICDLL	License Handler DLL is missing
EE_LICEXPR_STATUS_MISSING_LICUIDLL	License Handler UI DLL is missing

GetLastError

```
const err_ctx* (PRV_API* GetLastError) ()
```

Description

Returns the last error context. Pertains to the last error resulting from calling functions in this interface.

Parameters Defined

None.

EE_URS_INFO

```
typedef struct tagEE_URS_INFO{}
```

Information about a URS

unsigned cbsize	Size of structure
PRV_CHAR name[EEINFO_MAX_STRING]	Name of URS. EEINFO_MAX_STRING is 255 characters.
EE_URS_TYPE type	URS type

EE_URS_TYPE

```
typedef enum tagEE_URS_TYPE {}
```

Type of runtime objects. URS is short for updateable runtime system.

EE_URS_TYPE_AKS	Aladdin-defined runtime objects.
EE_URS_TYPE_PUBLISHER	Publisher-defined runtime objects.

EE_URS_ITF_INFO

```
typedef struct tagEE_URS_ITF_INFO{}
```

Information about a URS interface.

unsigned cbsize	Size of structure
PRV_CHAR name[EEINFO_MAX_STRING]	Name of Interface. EEINFO_MAX_STRING is 255 characters.

License Data Implementation

Implementing License Data (Feature String)

Example

The following is an SDK example that can be used as a reference to implementing the feature string inside your application. In order to use this example copy and paste it into a Console Application project and link it to the prv_sdk.lib.



The remark " **TO-DO: use License Data here**" indicated in red must be replaced by your code.

```

////////////////////////////////////
// Code sample for obtaining license data (usually used to store feature string)
// from product activation licenses.
//
// Copyright 2000-2005 by Aladdin Knowledge Systems Ltd.
//
// All Rights Reserved
//
// Permission to use, copy, modify and distribute this
// software and its documentation for any purpose is
// restricted according to the software end user license
// attached to this software.
//
// Any use of this software is subject to the limitations
// of warranty and liability contained in the end user
// license. Without derogating from the above said, Aladdin
// Knowledge Systems Ltd. disclaims all warranty with regard
// to this software, including all implied warranties of
// merchantability and fitness. In no event shall Aladdin
// Knowledge Systems Ltd. be held liable for any special,
// indirect or consequential damages or any damages whatsoever
// resulting from loss of use, data or profits, whether in

```

```

// an action of contract, negligence or other tortious
// action, arising out of or in connection with the use or
// performance of this software.
//
// NOTES: this sample assumes that all the licenses in the license
//        expression are of type "product activation".
//
////////////////////////////////////

#include <iostream>
#include <vector>
using namespace std;

#include "prv_sdk.h"
#include "asm_license_api.h"

#define EC_MAX_PATH_LEN          260

// Request the license data from the license.
void GetLicenseData(ISDK* pSDK, SDKContext ctx, EE_LICEXPR_ELEM_INFO* lee)
{
    const ISDK_COMMUNICATION_PACKET ASMP = { AKS_PROTOCOL_NAME,
        ASM_SDKCMD_GET_ACTIVATION_LICENSE_DATA };

    unsigned out_length = 0;

    // See if there is any data.

    if(pSDK->TransmitToLicenseDLL(ctx, lee, (const unsigned char*)&ASMP,
        sizeof(ISDK_COMMUNICATION_PACKET), 0, &out_length))
    {
        // Create a buffer in the right size and zero it.

        vector<unsigned char> out(out_length, 0x00);
    }
}

```

```

    cout << "Data packet size is: " << out_length << endl;

    // Get the data.

    if(pSDK->TransmitToLicenseDLL(ctx, lee, (const unsigned char*)&ASMP,
    sizeof(ISDK_COMMUNICATION_PACKET), &out[0], &out_length))

    {

        ASM_ACTIVATION_LICENSE_DATA*    pALD =
        reinterpret_cast<ASM_ACTIVATION_LICENSE_DATA*>(&(out[0]));

        cout << "License data is '" << pALD->data << "'" << endl;

        // TO-DO: use License Data here.

    }

    else

    {

        cout << "Failed to get license data" << endl;

    }

}

else

{

    cout << "No license data was found for this license" << endl;

}

}

/*-----*/
-----*/

int main()

{

    char ec_path[EC_MAX_PATH_LEN+1] = {0};

    // Get the module full path

    GetModuleFileName(NULL,ec_path,sizeof(ec_path));

    ISDK *ec_sdk = NULL;

```

```
SDKContext sdk_context = NULL;

// Load the Hasp SL execution environment.
if( PRV_SDK_LoadExecutionEnvironment( TRUE))
{
    cout << "Hasp SL execution environment was loaded" << endl;

    // Load the Hasp SL SDK interface.
    if( PRV_SDK_LoadSDKInterface(ec_path,&ec_sdk))
    {
        cout << "Hasp SL SDK interface was loaded" << endl;

        // Initialize the Hasp SL SDK interface.
        if( ec_sdk->Initialize())
        {

            cout << "Hasp SL SDK interface was initialized" << endl;

            // Get the Hasp SL SDK context.
            if( ec_sdk->GetContext(ec_path,&sdk_context))
            {
                EE_LICEXPR_ELEM_INFO* expr = NULL;
                unsigned expr_len;

                // Get the current effective set length.
                if(ec_sdk->GetEffectiveSet(sdk_context,0,&expr_len))
                {
                    unsigned i;
                    expr = new EE_LICEXPR_ELEM_INFO[expr_len];

                    // Get expression size.
```

```
    for( i=0 ; i < expr_len ; ++i )
    {
        expr[i].cbsize = sizeof(EE_LICEXPR_ELEM_INFO);
    }

    // Get the effective set.
    if( ec_sdk->GetEffectiveSet(sdk_context,expr,&expr_len))
    {
        // Get the license data from ever item in the efective set.
        for( i=0 ; i < expr_len ; ++i )
        {
            GetLicenseData(ec_sdk,sdk_context,&expr[i]);
        }
    }
    else
    {
        cout << "could not get effective set" << endl;
    }
}
else
{
    cout << "could not get effective set length" << endl;
}
}
else
{
    cout << "could not get context, aborting" << endl;
    exit(1);
}

ec_sdk->Finalize();

cout << "Hasp SL SDK interface was finalized" << endl;
```

```
    }  
    else  
    {  
        cout << "could not initialize isdk interface, aborting" << endl;  
        exit(1);  
    }  
  
    PRV_SDK_UnloadSDKInterface(ec_sdk);  
    cout << "Hasp SL SDK interface was unloaded" << endl;  
}  
else  
{  
    cout << "could not load isdk interface for" << ec_path << ", aborting" << endl;  
    exit(1);  
}  
  
    PRV_SDK_UnloadExecutionEnvironment();  
    cout << "Hasp SL execution environment was unloaded" << endl;  
}  
else  
{  
    cout << "could not load execution environment, aborting" << endl;  
    exit(1);  
}  
return 0;  
}
```