Welcome

Concept Software created the core technologies of the **SoftwareKey System™** used by thousands of ISV's (Independent Software Vendors) and enterprise organizations around the globe for almost two decades with our technology running on more than 100 million desktops and servers. Our very first customers are still productively using the SoftwareKey System and we are quite proud of that.

We continually offer new features as we develop our products in direct response to our customer's needs and wants. We are developers ourselves so we truly understand how unique each of your needs can be when implementing **software licensing, license management, copy protection, e-commerce and metering solutions**.

Concept Software makes customer service and flexibility in our products a top priority because it ensures your success.

Instant Protection PLUS 3

Instant Protection PLUS 3, which may be purchased separately, provides a wizard-based interface, and requires little to no source code changes to add the most commonly used licensing and activation features to your application. If you are new to licensing or the SoftwareKey System™, you may want to try the **Instant Protection PLUS 3** wizard first, and get started in minutes! Read **THE DECISION: Instant Protection PLUS 3 or Protection PLUS 5 SDK - SoftwareKey Blog** for help in deciding what product best suits your needs.

Protection PLUS 5 SDK

Protection PLUS 5 SDK is a software licensing client, which includes various application programming interfaces (APIs) and supporting applications to streamline integration of licensing into your application while providing the highest level of flexibility.

Application Programming Interfaces (APIs)

PLUSManaged

PLUSManaged is a fully managed .NET class library, which provides a rich, object oriented licensing API.

PLUSManagedGui

PLUSManagedGui is a fully managed .NET component that supplements PLUSManaged, ands provides a visual component that provides a licensing GUI API using `System.Windows.Forms` dialogs. This component can save you a great deal of time that would otherwise be spent designing and implementing your own graphical user interfaces (GUIs) for license management features in your protected applications.

PLUSNative

PLUSNative is a native, C library which provides a rich licensing API for native/unmanaged applications (or applications not built with .NET).

Why use Protection PLUS 5 SDK?

Whether your application has some niche licensing requirements, you find the **Instant Protection PLUS 3** wizard is not suitable to your needs or requirements, or you simply prefer to have the highest level of flexibility and control; Protection PLUS 5 SDK offers robust, flexible APIs which can save you a great deal of time. Some of the many benefits you gain by selecting the Protection PLUS 5 SDK APIs are highlighted below.

- The **license file** format leverages XML, which allows for highest level of flexibility and extensibility.
- License files are **encrypted** to keep your licensing data private, while allowing your application to access decrypted data with ease.
- License files are **digitally signed** using the RSA Algorithm for verification of the integrity and authenticity of licenses issued.sn
- Store your application's **license files** in any location! For example, you can store it on the file system, in a database, or in any place best-suited for your needs.
- Seamlessly integrate with SOLO Server web services for **Electronic License Management** (ELM) and much more!
- Leverage the same **security** used with license files for your application's web service calls made with SOLO Server.
- A robust, extensible **system identification** system allows you to pick and choose from a variety of built-in identification algorithms, while also giving you the power to implement your own algorithms for uniquely identifying licensed systems.

Get Started Now!

Whether you are evaluating the SoftwareKey System™ for the first time, or you wish to reproduce your successes in additional applications, it is best to keep informed with **what's new** to ensure your application has access to the latest compatibility updates, fixes, and the latest and greatest features. Furthermore, you should familiarize yourself with how to **get help** when needed, and you should make sure the latest **system requirements** fit your application's needs.

New to Protection PLUS 5 SDK?

First, it is very important to understand the SoftwareKey System™ as a whole to understand how Protection PLUS 5 SDK functions as an important part of it. Especially if you are new to Protection PLUS 5 SDK, it is best to **read more about the SoftwareKey System™**.

Furthermore, Protection PLUS 5 SDK is a licensing toolkit that is designed by developers for developers. Consequently, this manual is relatively technical in nature, and anything in the "Using Protection PLUS in your application" section is tailored for developers. More specifically, using PLUSNative effectively requires at least a basic understanding of application development and procedural programming, while PLUSManaged requires a basic understanding of .NET application development and object-oriented programming (particularly concepts such as inheritance and polymorphism).

Additional Resources

The latest information is always available on the web at **www.SoftwareKey.com**. This makes **online references** available, which includes the latest product manuals and API references. Additionally, the **getting help** section points to other helpful resources such as a knowledge-base and forums.

This manual was last updated Sunday, July 9, 2017.

What's New in Protection PLUS 5 SDK

Each release of Protection PLUS 5 SDK has a variety of changes and enhancements. The full list of changes may be reviewed in detail in our **release notes**.

Protection PLUS unbundled

The vast majority of our Protection PLUS customers were using either Instant Protection PLUS (previously called Instant PLUS) or Protection PLUS SDK, though they were previously bundled into one product.

As of Instant Protection PLUS 3.3 and later, Instant Protection PLUS 3 and Protection PLUS 4 and 5 SDK are separate products.

There are several advantages for our customers with unbundling:

- Lower cost for annual maintenance
- Simplified version updates
- Eliminates non-essential tools

Don't worry, most existing Protection PLUS customers with active maintenance will be able to access both products using their existing license. You can read more about this change on our blog: **Two Products are Better than One - A Case for Unbundling Protection PLUS**.

Version 5.17.2.0

Protection PLUS 5 SDK Native Edition

- New! WPF MVVM sample. This sample demonstrates activating specific features of an application that utilizes a WPF UI with the MVVM standard modeling.
- New! Visual Studio 2017 solution and sample projects.
- New! Visual Basic .NET Cloud Controlled Network Floating Licensing sample.
- The `NetworkSession` class may now optionally push a computer name string to SOLO Server to be displayed when viewing the active sessions.
- Writing a license to an image file using steganography no longer alters any of the file dates. This makes it less obvious that Protection PLUS 5 SDK is writing to the image file containing a license.
- Samples show a warning in their license status displays when activated with a SOLO Server test license.
- Samples now validate the system identifiers before writing a license. This fixes an issue with writable licenses where copying the license from another computer and immediately refreshing the license would create a valid license. After the refresh retrieves the license from SOLO Server, the sample code would write this license with the current system identifiers which would allow the license to pass any further validations. The current system identifiers are now compared to the identifiers in the license file on the computer before overwriting it.
- Visual Studio 2005 and 2008 samples have been removed from the installation. These samples are available upon request.

Protection PLUS 5 SDK .NET Edition

- Important! A dependency was updated which may contain critical security updates.
  - Updated OpenSSL to 1.0.2l
- New! Visual Studio 2017 solution and sample projects.
- Fixed an issue when opening a semaphore file using SK_PLUS_NetworkSemaphoreOpen where if it failed for any reason it may have caused SK_PLUS_NetworkSemaphoreStatistics to return incorrect values.
- The SK_SYSTEM_IDENTIFIER_ALGORITHM_USER_NAME system identifier algorithm can fail on RHEL/CentOS7 Operating Systems. There is now a fallback to retrieve the needed information from USER if the current implementation fails.
- Fixed an issue where SK_PLUS_NetworkSessionPoll created a session file when not checked out which could cause poll sessions to return an Invalid Data error.

- SK_PLUS_NetworkSessionPoll now supports the SK_FLAGS_NETWORKSESSION_SKIPLOCKATTEMPT flag.
- Fixed an issue with the Cloud Controlled Network Floating Licensing samples where closing a session in the SOLO Server interface would correctly close the session in the client on the next poll, but the client's poll timer was never stopped. This would cause the client to attempt a poll at each polling interval and an error message would be displayed. The poll timers in the samples are now stopped correctly.
- Visual Studio 2005 and 2008 samples have been removed from the installation. These samples are available upon request.

Version 5.17.1.0

Protection PLUS 5 SDK .NET Edition

- The LicenseProvider sample has been changed to use the CustomData field of the license rather than using the TC Fixed Value field for setting the DesignTime and RunTime modes. This simplifies the sample and allows these two modes to be enabled or disabled by editing the SOLO Server license and refreshing the client license.
- Two new features have been added to the LicenseProvider sample to enable and disable the control when running in Debug mode or Release mode.

Protection PLUS 5 SDK Native Edition

- Important! Two dependencies were updated, each of which may contain critical security updates.
  - Updated OpenSSL to 1.0.2j (see below)
  - Updated libcurl to 7.52.1
- Important! In the previous release OpenSSL was updated to version 1.1.0c on Windows only. Due to an issue we discovered when shutting down the OpenSSL library we have rolled it back to version 1.0.2j until the OpenSSL contributors address the issue. Due to the new OpenSSL design, its static library should not be shut down programmatically, but instead shuts down automatically when the calling process exits. This can cause issues when closing a PLUSNative context as it calls the OpenSSL shutdown function, and a subsequent open of a PLUSNative context will use the OpenSSL library in an unstable state.
- Added x64 build configurations to the Cloud Controlled Network Floating Licensing samples.

***Note** that changes to the Native Edition libraries also apply to the Android Edition, Java Edition, and LabVIEW Edition.

Version 5.16.4.0

General

- The manual has been updated to reflect the changes in our **new SOLO Server authors administration interface**.

Protection PLUS 5 SDK .NET Edition

- New! Added a sample implementing **Microsoft's LicenseProvider for Components and Controls**. The sample demonstrates licensing a simple user control.
- Fixed an issue with `LicenseAliasValidation` where validating the aliases before the main license could cause the main license to update with the current system identifiers if no aliases are present and `WriteMissingAliases` is true. This now only creates the missing aliases if there is already at least one alias present; otherwise, an error is returned.
- The writable license samples now validate the license file after a refresh before it is written to the drive.

- ASP.NET samples now use a static class to retrieve needed URLs rather than store them in web.config. Previously, the web.config could be modified allowing an intruder to change the URLs. Now the URLs are internal to the code making it difficult to modify them.

Protection PLUS 5 SDK Native Edition

- Important! OpenSSL was updated to version 1.1.0c **on Windows only**. All other supported platforms use version 1.0.2j, and will be updated to use the 1.1.0 branch in the future.
- The Android sample has been converted to use Gradle for building.
- Fixed an issue where system identifiers with the same hash value could cause the number of matches found to be larger than the number of identifiers when using `SK_PLUS_SystemIdentifierCompare`. This could cause an error when using Cloud Controlled Network Licensing.

Version 5.16.3.0

General

- The manual has been updated to use responsive HTML5 output and also includes various content tweaks.

Protection PLUS 5 SDK .NET Edition

- New! Added high-DPI awareness to PLUSManaged and PLUSManagedGui samples.
- New! Added support for new `GetLicenseCustomData`, `GetLicenseCustomDataS`, `UpdateLicenseCustomData`, and `UpdateLicenseCustomDataS` methods of the **XMLLicenseService** SOLO Server web service.
- `LicenseCounter` and `IsTestLicense` properties added to the **InfoCheck** class to easily retrieve the values from the web service.
- Fixed an issue where activating the sample applications manually with trigger codes did not save the Product ID, and caused a validation error to be displayed.

Protection PLUS 5 SDK Native Edition

- Important! Two dependencies were updated, each of which may contain critical security updates.
  - Updated OpenSSL to 1.0.2j
  - Updated libcurl to 7.50.3
- New! Java sample tutorial in the manual demonstrating the step-by-step use of our sample application.
- New! Proxy support added to all samples with the exception of Android and LabVIEW samples.
- Fixed small memory leak caused by attempting to open a License file that does not exist.
- All wxWidgets samples have been updated to use wxWidgets 3.1.0.
- Fixed debug warnings in the wxWidgets samples caused by invalidly combining some flags as well as using a few deprecated font enumerations.
- Fixed a text length limit with the MFC samples that would not allow manual activations when the response text was too large due to the CEdit control's default text length.
- Corrected the entry for SK_PLUS_LicenseAliasGetValidatedCount in the PLUSNative.bas file.

***Note** that changes to the Native Edition libraries also apply to the Android Edition, Java Edition, and LabVIEW Edition.

Getting Help

Online Resources

Our **support center** offers variety of resources that makes valuable information available to you immediately. This includes resources like a knowledge-base, forums, product documentation, newsletters, and more. Usually these resources give you near-instant answers to the questions and/or issues you encounter, and can therefore save you a great deal of time.

Getting Assistance

Evaluations

If you are new to our system and are evaluating our products and services, you will have access to guidance via email support during your evaluation period to help ensure you are acquainted and confident with our products and services.

Maintenance Subscriptions

Technology is constantly changing, and though this brings much excitement, it is understood that changes often result in unforeseen problems. Furthermore, keeping up with changes in various platforms, architectures, and frameworks requires a great deal of effort. Customers with active maintenance subscriptions have access to the following benefits for the products and services purchased:

- The latest compatibility updates around constantly-changing operating systems and development tools. Keeping up-to-date with the latest versions of our libraries minimizes the risk of encountering bugs and incompatibilities as things change.
- Email technical support.
- The latest and greatest improvements and features we are constantly adding to our products and services.

To check the status of your subscription, renew it, or download version updates, please sign-in to our **customer license portal**.

Professional Services

If your needs exceed that of our standard technical support services, we offer personalized professional services which may be used to further help you with an implementation or diagnosis. Should you find yourself in need of such services, please visit our **support center** and open a ticket to submit your request.

Gathering Information

You encountered a problem, could not find any information about the problem in our **support center** resources, and you need some answers quickly. Our products and services are designed to make life in the licensing and e-commerce world easier, and we take pride in fulfilling that same goal with technical assistance. However, you can make our lives a little easier and get answers much more quickly by taking the time to ensure your request includes all of the information we need to provide you with the answers you need. Here are some common questions we might need you to answer:

Behavior

- What behavior was experienced, and how does it differ from the behavior expected?
  - Can this behavior be reproduced using a sample application which was shipped with the product?
  - If any error/exception messages were displayed, please include the exact text and/or a screen shot.

- Who experienced the problematic behavior?
  - Did you, one of the application developers, and/or one of your customers experience the problematic behavior?
  - Is there a discrepancy between who experienced the behavior versus who should have (or should not have) experienced the behavior?
- When does this behavior occur?
  - Does the behavior occur consistently or intermittently?
  - Does the behavior occur from a specific line of code, area, and/or function or method call in your application?
- How can we reproduce this behavior?
  - Explain the steps taken to reproduce this behavior.
  - Include any screen shots if available.
  - If possible, provide a complete sample project (with source code) which can be used to reproduce the behavior.

Environment

- Did the affected systems belong to someone within your organization, the customer, or both?
- On what operating system was the behavior experienced? Please include the operating system version and architecture (i.e. Windows 7 Professional x64).
- Is there anything known about the environment which is unique, which might cause adverse affects? (i.e. does the problem only happen when a specific device is attached/detached, is it only on a laptop but not a desktop, etc...).

Application

- What programming language was used to write your application?
- What architecture (i.e. x86/32-bit, x64/64-bit) does your application target?
- Which of our tools or APIs (i.e. Instant Protection PLUS 3, PLUSManaged, PLUSManagedGui, PLUSNative) is used in your application? Are you using the latest version, and if not, what version does it use?
- Have any recent changes been made to the application? This can include things like: source code changes, using a different version of our tools or APIs, using a different version of development tools/frameworks,etc...

Submitting Inquiries

To submit your request for assistance, please visit our **support center** and open a new ticket. When doing so, making sure you include any relevant information (including, but not limited to, items listed in the "gathering information" section above). Additionally, if you have multiple inquiries or issues that are unrelated to one-another, open separate tickets for each inquiry/issue.

System Requirements

General Development Requirements

A screen resolution of 1024x768 pixels or larger is required for sample applications, documentation, and supporting applications. A higher screen resolution is necessary for text sizes larger than the default (96 pixels-per-inch in Windows).

PLUSManaged Requirements

Runtime

PLUSManaged requires the Microsoft .NET Framework 2.0 or higher to operate, and consequently, the PLUSManaged library shares the system requirements with the version of the Microsoft .NET Framework with which it is used. The Microsoft .NET Compact Framework, versions of the Microsoft .NET Framework that target mobile platforms and devices, and versions of the Mono Framework that target mobile platforms and devices are not supported.

If your application uses the Mono framework, PLUSManaged will operate under Mono versions 2.8 or later.

Important

When using PLUSManaged in Mono, and especially in platforms other than Microsoft Windows, it is important to note that certain features are not supported. This includes:

- Any methods or objects that rely on accessing the Windows registry (such as `LicenseWindowsRegistryAlias` and `IOHelper.ToUncPath`).
- Any methods or objects which rely on Windows Management Instrumentation (WMI) queries (includes the `WmiFixedOnly` filter in `HardDiskVolumeSerialIdentifierAlgorithm` and `VirtualMachineValidation`).
- Time validation via Simple Network Time Protocol (SNTP).
- Permissions cannot be set automatically on writable license files, so this is especially important to address during installation of your application on non-Windows environments.
- Mono's SSL/TLS support is limited and fails frequently depending on the server's cryptographic requirements. However, any calls made from PLUSManaged will automatically fall-back to plain HTTP.

Important

When using Network Floating Licensing features (via the `NetworkSemaphore` class) with protected applications, Windows Vista/Server 2008 or later is required for both the client computers accessing the protected application, and the file server hosting the share where the semaphore files will be stored.

Development

Visual Studio 2005 or later (with the latest service packs and updates available) is required when developing for the Microsoft .NET Framework. Xamarin Studio or MonoDevelop 2.8 or later is required if developing for the Mono framework.

Important

When developing with Visual Studio 2010, Service Pack 1 is required.

Using the latest service pack and updates available for the applicable version(s) of Visual Studio is strongly recommended.

PLUSManagedGui Requirements

Runtime

PLUSManagedGui requires the Microsoft .NET Framework 2.0 or higher to operate, and consequently, the PLUSManagedGui library shares the system requirements with the version of the Microsoft .NET Framework with which it is used. The Microsoft .NET Compact Framework, and versions of the Microsoft .NET Framework that target mobile platforms and devices are not supported.

A minimum screen resolution of 800x600 is required, though 1024x768 is the recommended minimum. A higher screen resolution is necessary for text sizes larger than the default (96 pixels-per-inch in Windows).

Important

Using PLUSManagedGui in the Mono framework is not supported.

Development

For development environments, Visual Studio 2005 or later (with the latest service packs and updates available) is required, and Visual Studio 2010 or later is recommended for PLUSManagedGui.

Important

When developing with Visual Studio 2010, Service Pack 1 is required.

Using the latest service pack and updates available for the applicable version(s) of Visual Studio is strongly recommended.

PLUSNative Requirements

An x86 compatible processor is required for 32 bit applications, or an x86_64 compatible processor is required for 64 bit applications.

Runtime

Windows

- Windows XP or later with the latest service packs and updates.

macOS

- Snow Leopard (10.6) or later with the latest updates.

Linux

- Linux 2.6 kernel or later.
- libc 2.5 or later.

Popular distributions supported include:

- CentOS 5.5 or later.
- Debian 5.0.8 or later.
- Fedora 13 or later.
- Madriva 2010.2 or later.

- OpenSUSE 11.2 or later.
- RedHat Enterprise Linux 5.6 or later.
- Slackware 13.1 or later.
- Ubuntu 8.04 or later.

Development

Windows

- Windows XP or later with the latest service packs and updates, with .NET Framework 4 (either Client Profile or the full version) installed.
- Static libraries require Visual Studio 2005 or later with the latest server packs and updates, though Visual Studio 2008 or later is strongly recommended. Using any PLUSNative libraries with Visual Studio 2005 requires a **hot-fix from Microsoft**.
- Dynamic link libraries (DLLs) require Visual Studio 6.0 or later with the latest service packs and updates is required. Visual Studio 2008 or later is strongly recommended.

macOS

- A 64 bit (x86_64) compatible processor is required.
- Leopard (10.6) or later is required, Lion (10.7) or later is strongly recommended.
- The latest operating system and XCode updates are always strongly recommended.
- The Objective C samples require OS X 10.6 and Xcode 4.2 or later.
- GCC 4.0 or later is required.

Linux

- Linux 2.6 kernel or later.
- libc 2.5 or later.
- GCC 4.0 or later.

Important

Any software using 32 bit `time_t` values, including 32 bit Linux and macOS applications and libraries, may fail when using dates later than January 18, 2038, or dates earlier than December 13, 1901.

With versions 8.0 (Visual Studio 2005) and later of Microsoft Visual C++ compilers, 32 bit Windows applications use 64 bit time_t values. If your software is compiled using a different or older compiler or C runtime library, then your 32 bit software may be subject to the limitations noted above.

Development System Recommendations

Some recommendations for your development systems are below. These tools are not required, and do require an initial investment in time and effort (particularly if you do not already use these tools); however, these tools can save you a great deal of time and headache.

- A **Version Control System** (VCS) is strongly recommended, as it allows you to keep track of changes made to source code and configuration files. This type of tool can provide a great deal of benefits even to a developer working alone on an application, and is usually a necessity when multiple developers collaborate on an application. When committing or checking-in changes to your version control system, using verbose comments about what is being done and why it is being done can often be very helpful in the future. This can be very helpful and important, even with the smallest of changes. Version control systems are typically very easy to create and configure, often taking only a few minutes to download, install, and configure.
- An **Issue tracking system** can also be very helpful for tracking past and current issues and enhancements, and can help you manage priorities between releases. Many issue tracking systems also have features that assist with

project management, which can be leveraged to delegate tasks amongst a group of developers, establish a work-flow amongst individuals, and much more. Additionally, many version control systems can integrate with issue tracking systems so that a change or set of changes can be linked with an issue.
- A **file comparison** (or diff) utility is very useful when reviewing changes made to your source code, and when troubleshooting issues that arise after some source code changes are made. Some version control systems may install a file comparison and merge utility.
- **Backups** (and at least some basic form of disaster recovery) are very important! Here are some considerations for your backups:
  - Backup any source code and version control system data (which would include the source code, and all revisions made to the source code). Make sure your source code backups include a backup taken before modifying your code to add licensing, and backup changes made during and after applying changes for licensing.
  - If you are using an issue tracking system, back up its database regularly.
  - Backup the Protection PLUS 5 SDK installer that you downloaded.
  - If you are given a License Manager envelope, it is **critical** for you to back this up to a secure and reliable location!
  - Consider storing backups off-site regularly. There are several cloud storage and backup services that offer relatively low-cost solutions and peace of mind. If cloud backups are too costly or impractical for your needs, storing physical media (such as a tape backup, hard drive, or thumb-drive) at another location (even if it's something like a safe deposit box) can help protect your code and business from natural disasters.
- **Virtualization software** is helpful for **testing your protected applications**.

System Settings and Policies

There are some system and domain policy settings, which seem to only be encountered on rare occasion, that can prevent Protection PLUS 5 SDK cryptographic routines from functioning. In particular, security policies which affect what algorithms may be used with the Windows Crypto API can impact the Protection PLUS 5 SDK APIs (PLUSManaged and PLUSNative) as well.

One example would be if a system's or domain's policy was configured to strictly adhere to a FIPS 140 standard, which only allows the use of very specific cryptographic algorithms. Problems can arise when strictly adhering to older versions of the standard (such as FIPS 140-1), which prevent the use of newer, stronger algorithms which have been approved by NIST in later versions of the standard. Protection PLUS 5 SDK presently requires FIPS 140-2 with Annex A so that it may leverage strong cryptographic algorithms. Additionally, even though PLUSManaged uses FIPS 140-2 approved algorithms, it is not able to use certified implementations for all algorithms while maintaining compatibility with earlier versions of the .NET Framework (2.0 and 3.0). The standard version of PLUSManaged will consequently not function properly on Windows when restricting the system to only use certified implementations. However, a special build of PLUSManaged has been added as of version 5.14.2.0, which is the Library\FIPS folder in the installation directory. This special version does use FIPS certified implementations of the required encryption algorithms, but only supports .NET Framework version 3.5 and later.

Installed Components

Protection PLUS 5 SDK will install a number of components on your development systems. This summary outlines the components that may be installed, and also supplements the license agreement by categorizing components by the type of license granted. Note that each file name listed below may represent more than one version or build of a file with the given name, as different versions/builds are necessary when supporting different operating systems, processor architectures, etc...

Design Time Developer Tools and Components

Design time developer tools and components include content which is installed on licensed computers used for development, but are not redistributed with or as part of your application.

Description	File Names
Protection PLUS 5 SDK Installers and Uninstallers	• PLUSNet_Setup.exe • PLUSNet_Uninstall.exe • PLUSNative_Setup.exe • PLUSNative_Uninstall.exe
Protection PLUS 5 SDK launcher/dashboard	• ProtectionPLUS5.exe
Manuals and Documentation	• PLUS5.chm • PLUS5.pdf • PLUSManagedAPI.chm • HelpContentSetup.msha • PLUSManagedAPI.cab • ReleaseNotes_ ProtectionPLUS5.html
XML documentation files (used for enhanced Visual Studio Intellisense information)	• PLUSManaged.xml • PLUSManagedGui.xml
Visual Studio extensions and packages	• PLUSManagedGui.vsix

Run Time Distributable Components

Run time distributable components include content which may be distributed with or as part of your protected applications. See the **PLUSManaged deployment** and/or **PLUSNative deployment** topics for additional details on what may need to be redistributed with your application(s).

Description	File Names
PLUSManaged library and supporting assemblies	• PLUSManaged.dll • PLUSManaged.XmlSerializers.dll

PLUSManagedGui library	• PLUSManagedGui.dll
PLUSNative shared objects and dynamic link libraries (DLLs)	• libPLUSNative.dylib • libPLUSNative.so • PLUSNative.dll
PLUSNative static libraries and DLL import libraries (**NOTE** that since these files are statically linked as part of your protected application/binary, these individual files **must not** be redistributed.)	• libPLUSNative.a • PLUSNative.lib • PLUSNative.lib • PLUSNative_md.lib • PLUSNative_dllimport.lib

Online References

Error & Result Codes

References are available online for **SOLO Server result codes** and **client-side result codes**.

Product Manuals & API References

The latest API references and manuals are available online, and you may use the links below to go to or bookmark them:

- **Protection PLUS 5 SDK Manual**
- **PLUSManaged API Reference** (includes reference material for PLUSManagedGui)
- **PLUSNative API Reference**
- **SOLO Server Manual**
- **Instant Protection PLUS 3 Manual**

When you install Protection PLUS 5 SDK on a computer, this manual is installed on your computer, and can be found in your start menu, under *SoftwareKey Licensing System/Protection PLUS 5 SDK/Manual*.

Trademark Acknowledgments

Concept Software, Inc.

- SoftwareKey
- SoftwareKey System
- Protection PLUS
- PLUSManaged
- PLUSManagedGui
- PLUSNative
- Instant PLUS
- Instant Protection PLUS 3
- Instant SOLO
- Instant SOLO Server
- SOLO Server
- Electronic License Activation
- ELA
- Electronic License Management
- ELM

Apple Inc.

- Apple
- Mac
- OS X
- macOS
- Objective-C
- Xcode

Microsoft Corporation

- Microsoft
- Windows
- Win32
- Visual Basic
- Visual Basic .NET
- VB.NET
- Visual C#
- C#
- Visual C++
- Visual Studio
- ActiveX
- .NET
- .NET Framework

RSA Security, Inc.

- RSA
- RSA Algorithm

The Eclipse Foundation

- Eclipse

Software License Agreement

Last Modified: February 6, 2015

Please read the following Software License Agreement for software products (hereinafter "Software") provided by Concept Software, Inc. The definition of Software in this license agreement includes any updates, modification, bug fixes, upgrades, enhancements, or other modifications.

This is a LEGAL AGREEMENT between YOU and CONCEPT SOFTWARE and by clicking "I AGREE," YOU are agreeing to be bound by the terms of this License Agreement. If you do not wish to accept the terms of this License Agreement, do not click "I AGREE." Electronically downloading and using this software program is only allowed by those who accept this License Agreement. Therefore, your downloading and use of this software program is conditioned on your acceptance of the terms and conditions of this License Agreement. If you do not wish to accept this License Agreement, do not use the software program and promptly (within 30 days of purchase) return it for a full refund.

1. NO TRANSFER OF OWNERSHIP

A. No title to or ownership of the Software, and associated manuals and documentation, or to any copyright, trademark, trade secret or other proprietary intellectual property rights to the Software, and associated manuals and documentation, is transferred to you under this Agreement, except as explicitly authorized in this Agreement.

B. Any copyright or trademark notices, product identification marking or description, or notices of proprietary restrictions in the software or documentation may not be removed or altered, including, but not limited to, the following trademarks and logos: The SoftwareKey System(TM), Protection PLUS(TM) (PLUS), Instant PLUS, Instant Protection PLUS, SOLO Server(TM) (SOLO), Instant SOLO Server(SM) Electronic License Activation(TM) (ELA), and Electronic License Management(TM) (ELM) logos.

C. All rights not respectfully granted to you under this Agreement are reserved to Concept Software, Inc.

2. LICENSE GRANT

One or more of the following license grant(s) will apply to you depending on which licenses you purchase from Concept Software, Inc. To determine which license grant(s) apply to you, refer to the matrix below and also review the list of licenses describing your permitted use of the Software by going to **http://www.softwarekey.com**, choosing Customer Login at the top, and logging into your customer account. From this portal, you can view and print license keys and invoices, download Software updates, and purchase Software upgrades and maintenance renewals.

Product Name	License(s) Granted to You
Protection PLUS	2(A), 2(B)
Instant Protection PLUS	2(A), 2(B)
SOLO Server	2(C)
SOLO Server Lite	2(C)

Regardless of whether one or more of the following license grant(s) found in Sections 2(A), 2(B), or 2(C) below apply to you, you agree to the remainder of this Agreement in its entirety.

A. Design Time Developer Tools and Components: Concept Software, Inc. hereby grants you a non-exclusive, non-transferable license to use this Software, including but not limited to developer specific components which shall remain on your development computer. Refer to the license in your customer account to determine which Software

edition(s) were purchased, the number of licenses purchased, and for which operating system(s). You may install and use the Software on the number of development workstations for which you have a license. Refer to the license in your customer account to determine which Software edition(s) were purchased, the number of licenses purchased, and for which operating system(s) and/or platform(s).

B. Run Time Distributable Components: When run-time tools are included in your software, Concept Software, Inc. hereby grants you a non-exclusive, transferable, royalty-free license to use the Run Time Distributable Components ("Run-time Tools") which are identified in the user's manual, which may be included as part of your software developed with this Software. This non-exclusive, transferable, royalty-free license is limited to just your software and not to any secondary software created by your software.

C. Server Tools and Components: Any identified server tools and components may be installed on a server operating system used to provide other computers with web interface and/or license validation, activation, and update services without additional royalties. The Software may not be used to license applications created by another entity so as to compete with Concept Software, Inc. Refer to the license in your customer account to determine the number of Internet domain name(s) / URL(s) which may be used with the server and how many computers or virtual machines may use the Software.

Except for the licenses granted herein, you further shall not rent, lease, distribute, sell, or create derivative works of this Software. You may not adapt, translate, reverse engineer, de-compile, disassemble, modify, or otherwise attempt to discover the source code of the Software in whole or in part. Except as explicitly authorized in this Agreement, Concept Software, Inc. owns and retains all right, title, and interest in the Software, manuals, Documentation, and any and all other related materials you are not acquiring any rights of ownership in the Software, manuals, Documentation or any and all other related materials.

All rights not respectfully granted to you under this Agreement, including but not limited to license rights and intellectual property rights, are reserved to Concept Software, Inc.

3. PROHIBITIONS AGAINST INTELLECTUAL PROPERTY RIGHTS

A. The Software, and associated manuals and documentation, were each independently created and are protected by U.S. copyright laws and international treaty provisions. They may not be copied, in whole or in part, except as authorized in this License Agreement.

B. You shall not remove, obscure, or alter any notices of copyright, trademark or other proprietary notice or legends appearing in or on any Software, and associated manuals and documentation, of Concept Software, Inc., and shall reproduce all such notices on all copies of the Software, and associated manuals and documentation, as it appears. You acknowledge and agree any Software licensed to you by Concept Software, Inc., associated manuals, associated documentation and associated database structure contain copyrighted material which is owned exclusively by Concept Software, Inc.

C. The Software, and associated manuals and documentation, may not be copied except that one (1) copy of the Software, and associated manuals and documentation, may be made for backup or archival purposes. Any other copying not expressly authorized by this Agreement is prohibited.

D. Any threatened violation or actual violation of this Agreement is agreed to be a willful violation of the rights of Concept Software, Inc. Any threatened violation or actual violation shall result in irreparable harm to Concept Software, Inc. and shall entitle Concept Software, Inc. to an injunction as well as to other legal remedies, including an award of attorney's fees and costs.

4. TERM

Unless expressly stated otherwise, as long as the Software is used by you and/or is in your possession, the terms of this Agreement shall survive termination of this Agreement.

5. RIGHTS ON TERMINATION

Concept Software, Inc. has and reserves all rights and remedies that it has by operation of law or otherwise to enjoin the unlawful or unauthorized use of Software or Documentation. On termination, (a) all rights granted to you under this Agreement cease and you will promptly cease all use and reproduction of the Software and Documentation and (b) you will promptly return all copies of the Software to Concept Software, Inc. or destroy all of your copies of the Software and so certify to Concept Software, Inc. in writing within fourteen (14) days of termination. Sections 2, 8, 9, and 10 will survive termination or expiration of this Agreement as will any cause of action or claim of either party, whether in law or in equity, arising out of any breach or default.

6. TRIAL EDITION LICENSE

When evaluating components of the Software, Trial Editions may be used in a development environment for an evaluation for no longer than the specified evaluation period (typically determined by the date the evaluation of the Software is requested). The Trial Editions may only be used for evaluation, and redistribution of any trial libraries or protected applications is prohibited.

7. U.S. GOVERNMENT LICENSE GRANT - RESTRICTED RIGHTS

The Software, manuals, documentation, and other materials were developed at private expense by Concept Software, Inc., PO Box 770459, Winter Garden, FL 34777. Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in the Federal Acquisition Regulation (FAR 12.212; 27.400 through 27.409; and 52.227-19) and other U.S. Government agency acquisition regulations such as Department of Defense Federal Acquisition Regulations (DFARS 252.227-7015; and 227.7202), as applicable. The license grant to the U.S. Government shall be the same as that customarily provided to the public unless inconsistent with Federal procurement law.

8. LIMITED WARRANTY AND REFUND POLICY

If you are not happy with the Software, you have within thirty (30) days from purchase to (1) uninstall the Software, (2) destroy the manual and all other documentation and materials provided by Concept Software, Inc., and (3) email Concept Software, Inc., through **http://www.softwarekey.com/contact/**, to confirm these steps have been accomplished and to request a refund. Concept Software, Inc. shall provide you a 100% return on your purchase price after such verification is received, but only if completed and received within 30 days of purchase. There is no refund, or any value, otherwise.

THERE ARE NO REFUNDS FOR SERVER PRODUCTS SUCH AS SOLO LITE OR SOLO SERVER.

THE WARRANTIES SET FORTH DIRECTLY ABOVE, ARE IN LIEU OF, AND THIS AGREEMENT EXPRESSLY EXCLUDES, ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, ORAL OR WRITTEN, OR STATUTORY OF ANY KIND OR NATURE WHATSOEVER INCLUDING, WITHOUT LIMITATION, (a) ANY WARRANTY THAT THE SOFTWARE IS ERROR FREE, WILL OPERATE WITHOUT INTERRUPTION, OR IS COMPATIBLE WITH ALL EQUIPMENT AND SOFTWARE CONFIGURATIONS; (b) ANY AND ALL IMPLIED WARRANTIES OF MERCHANTABILITY; (c) ANY AND ALL WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE; (d) ANY AND ALL WARRANTIES OF NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS, AND (e) ANY AND ALL WARRANTIES OF TITLE AGAINST THIRD PARTY INFRINGEMENT.

9. LIMITATION OF LIABILITY

A. Concept Software, Inc.'s liability for damages resulting from use of the Software shall in no event exceed the amount of license fees paid by you to Concept Software, Inc. under this License Agreement. Concept Software, Inc. shall not in any case be liable for damages incurred by third party users of applications that incorporate Software and/or Software licensing functions. Your implementation of the Software is in no way influenced by Concept Software, Inc. its officers, employees or agents. The foregoing provision shall be enforceable to the maximum extent permitted by applicable law.

B. CONCEPT SOFTWARE, INC. SHALL NOT IN ANY CASE BE LIABLE FOR SPECIAL, INCIDENTAL, CONSEQUENTIAL, INDIRECT, OR OTHER SIMILAR DAMAGES (INCLUDING WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, REVENUE, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, COST OF PROCUREMENT OF SUBSTITUTE GOODS INCURRED BY YOU OR ANY THIRD PARTY), WHETHER IN AN ACTION IN CONTRACT OR TORT OR BASED ON A WARRANTY OR ANY OTHER PECUNIARY LOSS, ARISING FROM THE USE OR INABILITY TO USE THE SOFTWARE, MANUAL, DOCUMENTATION, OR OTHER MATERIALS, EVEN IF CONCEPT SOFTWARE, INC. OR ANY OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

C. You agree to defend and indemnify Concept Software, Inc. and to hold it harmless from and against any and all claims, actions, proceedings, judgments, losses, liabilities, costs and expenses (including attorney fees and expenses) arising out of or relating to any of your activities or inaction and/or arising out of or relating to any actions or claims of your customers.

10. LIMITATIONS PERIOD

No action arising out of or in connection with this Agreement or the transactions contemplated by the Agreement may be brought by either party against the other more than spelled number of days 365 days after the action accrues.

11. BREACH OF AGREEMENT - TERMINATION

Any breach of one or more of the provisions of this License Agreement shall result in immediate termination of this License Agreement. You shall then immediately discontinue use of and return all copies of the Software, manuals and documentation to Concept Software, Inc. or supply a certificate of destruction of all copies. All provisions of this License Agreement that protect the rights of Concept Software, Inc. shall survive termination.

12. ASSIGNMENT

You may not assign, sublicense, or transfer your rights or delegate your obligations under this Agreement without Concept Software, Inc.'s prior written consent, which will not be unreasonably withheld. This Agreement shall be binding upon the successors and assigns of the parties to this Agreement.

13. EXPORT

You agree to abide by U.S. and other applicable export control laws and agree not to transfer the product to a foreign national, or national destination, which is prohibited by such laws, without first obtaining, and then complying with, any requisite government authorization. You certify that you are not a person with whom Concept Software, Inc. is prohibited from transacting business under applicable law.

14. MODIFICATION

This Agreement may not be modified or amended except in a writing signed by an authorized officer of each party.

15. SUPERSEDING AGREEMENTS

In the event that you have entered into a written agreement with Concept Software, Inc., which applies to the same product license(s) as this Agreement, negotiated separately from this Agreement, some terms of which are in conflict with the terms of this Agreement, the differing terms in that negotiated agreement shall prevail if, and only if, said written agreement expressly states that those differing terms supersede the terms in this Agreement. Otherwise, Section 23 of this Agreement applies.

16. WAIVER OF CONTRACTUAL RIGHT

The failure of Concept Software, Inc. to enforce any provision of this Agreement shall not be construed as a waiver or limitation of any rights of Concept Software Inc. to subsequently enforce and compel strict compliance with every provision of this Agreement.

17. GENERAL

You are responsible for compliance with all laws and regulations governing export outside the United States of any product containing the Software, such as obtaining and renewing licenses or permits and financial reporting obligations.

18. HEADINGS

The headings used are for convenience only.

19. APPLICABLE LAW

This agreement shall be governed by the laws of the State of Florida, without regard to conflict of law provisions.

20. ARBITRATION

In the event of any dispute between the parties arising out of this Agreement, the dispute shall be resolved by arbitration under the rules of the American Arbitration Association by an arbitrator agreed upon in writing by the parties. In the event the parties cannot agree upon the choice of an arbitrator, each party shall appoint one individual representative and the two party representatives shall, between themselves, chose an arbitrator.

21. THIRD-PARTY LICENSES

This Software includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org/). This Software includes cryptographic software written by Eric Young (eay@cryptsoft.-com).

22. SEVERABILITY

In case any provision of this Agreement is held to be invalid, unenforceable, or illegal, the provision will be severed from this Agreement, and such invalidity, unenforceability, or illegality will not affect any other provisions of this Agreement.

23. ENTIRE AGREEMENT

This Agreement is the entire agreement between you and Concept Software, Inc. relating to the subject matter of this Agreement and superseded all prior and contemporaneous understandings or agreements of the parties. This Agreement may not be contradicted by evidence of any prior or contemporaneous statements or agreements. No party has been induced to enter into this Agreement by, nor is any party relying on, any representation, understanding, agreement, commitment or warranty outside those expressly set forth in this Agreement.

Third-Party Licenses

The following components are used in the PLUSNative API. Since we redistribute these components with our libraries, we are required to include references to the license agreements in our documentation. Before redistributing PLUSNative with your application, you should review these licenses to see how their conditions affect your application and its license agreement (i.e. you may need to simply include these with your application's license agreement).

libcurl

libxml2

OpenSSL

```
/*
 ====================================================================
 * Copyright (c) 1998-2011 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 * software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
```

* Hudson (tjh@cryptsoft.com).
*
*/
Original SSLeay License

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
* All rights reserved.
*
* This package is an SSL implementation written
* by Eric Young (eay@cryptsoft.com).
* The implementation was written so as to conform with Netscapes SSL.
*
* This library is free for commercial and non-commercial use as long as
* the following conditions are aheared to. The following conditions
* apply to all code found in this distribution, be it the RC4, RSA,
* lhash, DES, etc., code; not just the SSL code. The SSL documentation
* included with this distribution is covered by the same copyright terms
* except that the holder is Tim Hudson (tjh@cryptsoft.com).
*
* Copyright remains Eric Young's, and as such any Copyright notices in
* the code are not to be removed.
* If this package is used in a product, Eric Young should be given attribution
* as the author of the parts of the library used.
* This can be in the form of a textual message at program startup or
* in documentation (online or textual) provided with the package.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions
* are met:
* 1. Redistributions of source code must retain the copyright
* notice, this list of conditions and the following disclaimer.
* 2. Redistributions in binary form must reproduce the above copyright
*   notice, this list of conditions and the following disclaimer in the
*   documentation and/or other materials provided with the distribution.
* 3. All advertising materials mentioning features or use of this software
*   must display the following acknowledgement:
*   "This product includes cryptographic software written by
*   Eric Young (eay@cryptsoft.com)"
*   The word 'cryptographic' can be left out if the rouines from the library
*   being used are not cryptographic related :-).
* 4. If you include any Windows specific code (or a derivative thereof) from
*   the apps directory (application code) you must include an acknowledgement:
*   "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
*
* THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
* ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
* FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
* DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
* OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
* LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
* OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.

```
*
* The licence and distribution terms for any publically available version or
* derivative of this code cannot be changed. i.e. this code cannot simply be
* copied and put under another distribution licence
* [including the GNU Public Licence.]
*/
```

SoftwareKey System Overview

The SoftwareKey System™ is a complete, turnkey solution for software licensing, online license management, automated online software activation, and optionally selling software online. This system is comprised of two products, which may be used independently, but yield the best results when used together. This seamless and powerful client-server combination is suitable for a one-person operation or a sophisticated team of developers. Designed with easy-to-use wizards and wrappers or, if you prefer, the access and flexibility for you to customize and integrate to your heart's content.

SOLO Server

SOLO Server is power packed with features including **Electronic License Management™** (ELM) allowing you and your customers to activate and manage their software 24 hours a day while you maintain control. ELM is the must-have luxury feature for software companies who want to automate their online software business, both with or without integrated e-commerce. SOLO Server is available as a service (SaaS) and as an on-premise solution. **Learn more** today, and check out the most popular **features**! We recommend using SOLO Server, but you can also use **License Manager** to manually activate customers offline.

Instant Protection PLUS 3

Instant Protection PLUS 3 is an alternative to the Protection PLUS 5 SDK application programming interfaces (APIs), which provides a wizard-based interface, and requires little to no source code changes. If you are new to licensing or the SoftwareKey System™, check-out **Instant Protection PLUS 3** to see just how easy it is to add licensing, activation, and optional e-commerce for your application within minutes!

Protection PLUS 5 SDK

Protection PLUS 5 SDK the industry standard, *software licensing* client which is designed to protect your investment in your intellectual property, and positions you to distribute and sell your software. This is distributed with your software, which runs on your customers' computers.

Content Protection

As a *software licensing* client, Protection PLUS 5 SDK is designed to protect software rather than arbitrary media or content (such as music, images, etc...) However, Protection PLUS 5 SDK is ideal for protecting applications which protect, manipulate, and/or produce content.

Application Programming Interfaces (APIs)

The Protection PLUS 5 SDK APIs offer a common way for your applications to create and manage licenses, and integrate with SOLO Server for **Electronic License Management™** (ELM). Using the APIs affords you and your application ultimate flexibility and control, while saving you an abundance of time by providing a wide range of licensing and ELM functionality in a single, easy-to-use package. They save you even more time by making this functionality available to you in a consistent manner, on a wide variety of platforms and development frameworks. For .NET developers, the rich, fully managed, object-oriented, PLUSManaged API provides an ideal experience. Likewise,

PLUSNative provides a rich procedural interface for developers writing native applications or applications which can call to shared/dynamic-link libraries. Whether you are new to Protection PLUS 5 SDK, or you are upgrading from Protection PLUS 4, take a look at **the differences** between these two generations to see how the SoftwareKey System™ has been evolving over the decades.

Protection PLUS 4 versus Protection PLUS 5 SDK

The previous generation, Protection PLUS 4, has been the licensing system of choice for thousands of software developers for well over 15 years. The initial design of Protection PLUS 5 SDK started in 2009 with the release of the PLUSManaged library for .NET, in use by many new and existing Protection PLUS 4 customers. Although Protection PLUS 5 SDK has been well received even by existing Protection PLUS 4 customers, migrating from Protection PLUS 4 to Protection PLUS 5 SDK is not seamless. Protection PLUS 5 SDK is an entirely new product built from the ground up from much of the same technologies and experiences that made Protection PLUS 4 powerful.

When we first set out to create Protection PLUS 5 SDK APIs, we had the following design goals in mind:

- Leverage all of the knowledge learned since Protection PLUS 4 was released, based upon feedback from customers
- Simplify the set of API interfaces available, and make the APIs easier to use and understand
- Combine the Automation Client (SKCAxx.dll), SWKClientServices, and Licensing Client (KEYLIBxx.dll) into a single library
- Make it easy to call SOLO Server web services from a standard API
- Use XML for the License File format for the greatest extensibility
- Use the RSA Algorithm to leverage public key cryptography and digital signatures to better secure the license file and its data
- Full support for UNICODE character sets
- Support additional platforms (such as macOS and Linux) and processor architectures
- Create the .NET component in 100% managed code to provide the greatest .NET project compatibility
- Create a secure activation process that eliminates the 14-bit limitation on user-defined data
- Provide the same features and increased **security** across all **activation** methods – online, email, activation from another computer that has Internet access
- Introduce new methods of licensing and activating, such as **volume licenses, and downloadable licenses with activation**

Comparison of Technology and Features

Feature	Protection PLUS 4	Protection PLUS 5 SDK
License File	Fixed-size, proprietary format, which cannot not always accommodate highly customized information.	Variable-size, XML format, easily accommodates highly customized information.
Cryptographic Strength	Proprietary, symmetric encryption algorithm.	Uses the RSA Algorithm for strong, scalable, asymmetric (public key) cryptography and digital signatures.
Activation	Based off an exchange of 32-bit numeric values (called Trigger Codes), can only accommodate a very limited license parameters, often called Payload.	Based off an exchange of encrypted and digitally signed XML blobs, and can accommodate a much larger payload. Also includes high-level support for processing Protection PLUS 4 activation codes (Trigger Codes).
System Identification	Uses up to 17 fixed pieces of information to authorize and verify licensed computers locally. Only a very small subset of this information is used during activation, however.	Includes a variety of algorithms designed to use any combination of data to authorize and verify licensed computers. Also supports defining your own algorithms, which enables you to use your own, customized sources of information for

		system identification. Additionally, the XML-formatted data is used during activation, which makes it much easier to tell what pieces of information changed, if needed. Although the number of algorithms available in Protection PLUS 5 SDK is limited (because of the extra effort required to support multiple platforms), this will be expanded over time.
Copy Protection	A fixed vote-weight system is used to detect changes made to a computer after activation. The amount of change allowed before a license is revoked is controlled by a numeric threshold value.	Discrete algorithms give you full control over which algorithms are used, and how many algorithms must match after activation.
SOLO Server Integration APIs	Includes the separate APIs, Automation Client (for native applications) and SWKClientServices (for .NET applications) for SOLO Server integration.	Everything you need for SOLO Server integration is now built-in with whichever Protection PLUS 5 SDK API you use.
Native API Operating System Support	Supports running 32-bit applications on 32-bit and 64-bit versions of Windows, and running 64-bit applications on 64-bit versions of Windows.	Supports Windows and other POSIX-compliant operating systems (such as macOS and most popular and current Linux distributions).
.NET APIs	Includes a 32-bit (only supports 32-bit applications, though 64-bit versions of Windows are supported), mixed-mode library named SKCLNET. This has some limitations as to where it can be used (due to limitations in the .NET framework with using strong named, mixed-mode assemblies).	Fully managed, .NET class library named PLUSManaged, which is architecture independent (targets "Any CPU"), and supports 32-bit and 64-bit operating systems and applications.
ActiveX/COM APIs	Includes a 32-bit (only supports 32-bit applications, though 64-bit versions of Windows are supported) ActiveX component named SKCL.	No ActiveX/COM library is presently available for Protection PLUS 5 SDK.
License File I/O	Every time a license file field is read, the entire license file is read from the file system. Every time a license file field is written, the entire license file is read from the file system, the field is updated, and then the entire license	The license file is only read and loaded in to memory, and is only written to the file system, when your application makes explicit API calls to do so. This improves performance in lower-concurrency environments, but also means you need to be mindful of this behavior and the

	file is written. This approach makes it easier to avoid synchronization issues in high-concurrency environments, but can also reduce performance (especially when the license file is stored in a shared folder on a network).	possibility of synchronization issues in high-concurrency environments. A major reason why this approach was taken is to minimize the amount of cryptography done when working with license file data, which especially important to minimize in high-concurrency/server environments.
Network Floating Licensing	Supported through the use of semaphore files hosted on a Windows shared folder on a network.	.NET applications using PLUSManaged support Network Floating Licensing through the use of semaphore files hosted on a Windows shared folder on a network. PLUSManaged
Cloud-Controlled Floating Network Licensing using SOLO Server	Supported through the use of SOLO Server web services that are used to manage the state of each license "session."	These web services may also be used to automate authorization of "check-outs" for temporary, off-line use of the licensed application. If you would like to use this functionality or see it demonstrated, **contact us**. These features are not yet available, but are planned for, native applications using PLUSNative.
Simplified APIs	Protection PLUS 4 includes EZTrial1, which simplifies licensing and trial-ware implementations; and EZTrial2, which is like EZTrial1 with a pre-defined GUI. EZTrial was replaced with the Instant Protection PLUS 3 licensing wizard.	Although there are no simplified APIs similar to EZTrial, Protection PLUS 5 SDK includes sample source code which is highly configurable, and makes it easy for you to adopt licensing and trial-ware functionality in your application, while also giving you the highest level of flexibility and control. Additionally, a pre-defined GUI is available for .NET applications via the PLUSManagedGui library.
License File Editor	LFEdit can be used to view, edit, and create Protection PLUS 4 license files. An additional utility (LFRW.exe) is included to view or edit some internal fields that are not visible in LFEdit. These applications only run in Windows.	Most customers use SOLO Server to create Protection PLUS 5 SDK license files, as part of an automated activation process. Alternatively, **License Manager** allows you to view, edit, and create both read-only and writable **license files** regardless of whether or not you are using SOLO Server. License Manager runs in Windows, macOS, and most popular Linux distributions. (See **system requirements** for details.)

Additional differences worth noting include:

- Protection PLUS 4 was introduced in 1997, while Protection PLUS 5 SDK was first released in 2009 with the PLUSManaged API. PLUSNative was added to the Protection PLUS 5 SDK toolkit in 2012.
- The Protection PLUS 5 SDK APIs are still evolving over time. There is not a 100% feature match between the Protection PLUS 4 and Protection PLUS 5 SDK, especially with the PLUSNative library.
- There is not currently a way to read a Protection PLUS 4 license file with the Protection PLUS 5 SDK libraries. We feel that in most cases, customers will want to switch to Protection PLUS 5 SDK in a major release of their software

where product re-activation would likely occur. However, it is possible to automate the **upgrade from Protection PLUS 4** to Protection PLUS 5 SDK for your customers licenses.

Cryptography and Security

Protection PLUS 5 SDK uses the latest in cryptographic technology to secure your application's license files and its communication with SOLO Server. Here, we provide you with a high-level understanding of the technology and how it is used. This helps you understand how Protection PLUS 5 SDK works, and how the technologies used benefit you.

The Role of Cryptography

The first and most important thing to understand about cryptography, and security in general, is that nothing is impervious to attack. With enough effort and/or resources, any system may be compromised. The purpose is to simply make it unreasonable for most people to take the time to successfully break or compromise a system and to keep honest people honest.

Furthermore, security generally needs to be balanced with accessibility/ease-of-use. An example of this is the option to use read-only **license files** (which must be issued by SOLO Server) and writable/self-signed license files (which may be written and signed by your application). The latter is less secure, but has advantages in certain scenarios.

Types of Cryptographic Algorithms

There are many different types of cryptographic algorithms, all of which have their own advantages and dis-advantages. Understanding the different types of algorithms used in Protection PLUS 5 SDK at a high level helps you understand how each one serves different purposes/functions.

Hash Algorithms

A hash algorithm is a one-way cryptographic algorithm (meaning it cannot be decrypted) that takes some data and generates a fixed-size "digest." Hash algorithms are ideal for tasks like authentication and integrity verification.

Authentication Example

Perhaps the most common use of hash algorithms is with authentication. Storing passwords in plain-text in a data-base or system which holds sensitive data is bad practice, but you need some kind of data to ensure the user entered a valid password. In this scenario, the solution is to store a password hash instead. Here is how this works:

- The user creates a new account in a system, and specifies "topsecret" (without the quotes) as his or her password.
- The system generates a digest using the SHA1 hash algorithm and stores the digest ("EiAf5eICiDvUX8l+hzZuoFGD4OQ=") instead of the password.
- When the user tries to sign-in later, he or she enters the password "topsecret" again. The system generates the SHA1 hash digest from the password entered, and checks to make sure it matches the hash stored earlier when the account was created.

Of course, the example above is a simple one, and is not perfect. If two users have the same password, they would have the same hash. In these scenarios, we use what is known as a "salt", which helps make the hash unique. Let's say the user enters a unique username with the password for authentication. As an example, one could add (or concatenate) with the password to act as the "salt." In other words, "bobtopsecret" would result in a different hash digest from "alicetopsecret."

Integrity Verification Example

Integrity verification is not much different from the authentication shown above. As an example, let's say you went to download the latest FreeBSD operating system ISO file. In the FTP folder you see from the web site, you notice there are some CHECKSUM files present. Here is how these work:

- Someone makes the FreeBSD ISO file and posts it to the FTP server.
- That same person then makes the hash digest/checksum file and posts that to the FTP server with the ISO file.
- You download the ISO and checksum files.

- Now you can generate a hash on the ISO using the same hash algorithm, and make sure it matches the digest in the checksum file to ensure your download is not corrupted or different in any way.

How Protection PLUS 5 SDK Uses Hash Algorithms

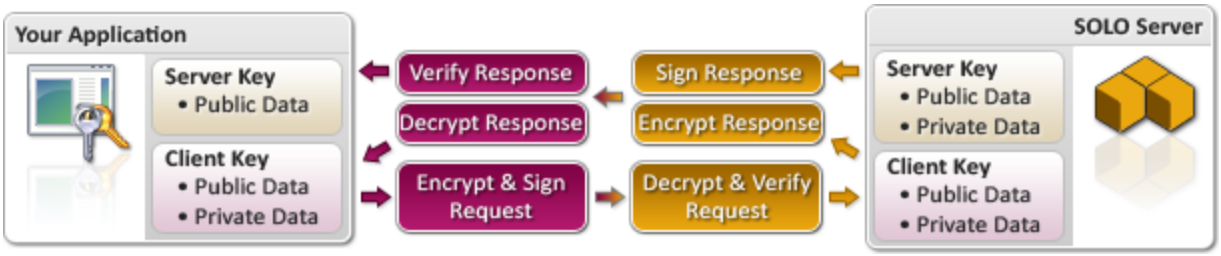
Protection PLUS 5 SDK leverages hash algorithms to:

- Store and validate computer/system information. By using hashes with this information, we can:
  - Verify the system information authorized in the license file.
  - Transmit the hashed system information over the Internet in a manner which does not violate the user's privacy, which is similar to how hashes are used in the authentication example above.
- Verify the integrity of the License File and web service communications (hashes are used as part of the digital signatures, which are explained later), which is similar to the integrity verification example above.

Asymmetric Algorithms

Asymmetric cryptography, also known as "public-key" cryptography, is where two keys are used with your application. This type of cryptography is widely used, and perhaps the best example is that your web browser uses this type of cryptography to encrypt data you access from and send to web sites (SSL). The drawback to this type of encryption is that it generally requires more resources from your computer to use than symmetric algorithms, but that is also what gives these algorithms their strength. In other words, the longer it takes to encrypt and decrypt data using an algorithm, the more impractical it becomes to guess the key used by an application by simply trying all possibilities.

The real strength in these types of algorithms, however, is that two separate keys (also referred to as a "key pair") are typically used. A flowchart showing how the SoftwareKey System uses this type of cryptography is below, and illustrates how your application only knows part of the server key (it does not know the server key's private data). The result is that your application only has enough of the server key to verify that it has been signed with the correct private key data, but lacks the key data necessary to sign data itself.



Using activation as an example, here is what happens:

1. Your user wants to activate. Your application uses one of the Protection PLUS 5 SDK APIs or Instant Protection PLUS 3 to generate a request, and encrypt & sign the request using the private data of the "Client Key".
2. SOLO Server receives the request, and first decrypts & verifies the request using the "Client Key."
3. SOLO Server processes the request, generates a response, uses the private data of the "Server Key" to sign the response, and uses the "Client Key" to encrypt the response.
4. Your application receives the response from SOLO Server, decrypts the response using the "Client Key," verifies the response using the public data of the "Server Key," and processes the response. In the case of activation, the response generally contains the license file, which your application decrypts & verifies in the same manner (for read-only license files).

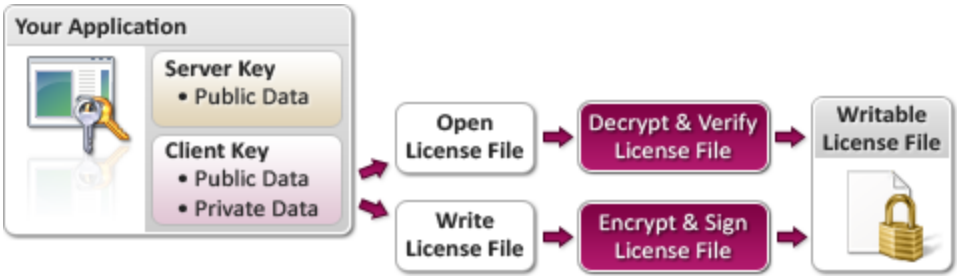
As you can see from the chart above, your application does not have access to the private data of the "Server Key." Since this data is not even available to your application, it is difficult to compromise the key in a way that allows a hacker to generate their own, arbitrary responses and/or license files. Of course, all of the cryptographic operations performed by your application are handled for you through one of the Protection PLUS 5 SDK APIs (PLUSManaged or PLUSNative).

Symmetric Algorithms

Symmetric cryptography, also known as "secret-key" cryptography, is where a single cryptographic "key" is used to encrypt and decrypt data. This is often used where performance is critical. The disadvantage to symmetric algorithms is that the key must be known to all parties, or in the case of the SoftwareKey System, the same key data must be fully disclosed to both SOLO Server and your application. If the "secret-key" is compromised (meaning someone finds it and publishes or uses it without authorization), then the security is compromised.

Writable License Files

Protection PLUS 5 SDK does make it possible to use "writable **license files**." These license files offer you an added level of flexibility for your application, as it enables your application to write to and sign license files itself. In this scenario, only the Client Key is used for all cryptographic operations, which means it operates like a symmetric or "secret-key" algorithm.



Even though the writable license is encrypted and signed by your application, measures should be taken (such as using an obfuscation tool if you code in .NET) to hide and protect the Client Key so that hackers cannot use it to write their own license files for your application.

Steganography

Important

The steganography feature is presently a beta, and changes may be applied which could require minor source code modifications to be made in order to use a future release. Please keep in mind that **your testing** should be very thorough. Also, use your own images distributed with your application such as a splash screen image. Do not use an existing image such as the desktop wallpaper since another product could share that license location.

Steganography is the practice of concealing data within a file, image, or video. Protection PLUS 5 SDK allows the use of steganography to conceal a license within a **supported image format**. When the license is written within the image, the image is still valid and may be viewed without any noticeable changes or degradation to the image data. Both PLUSManaged and PLUSNative support stenography for storage of licenses within images.

Philosophy regarding Back Doors

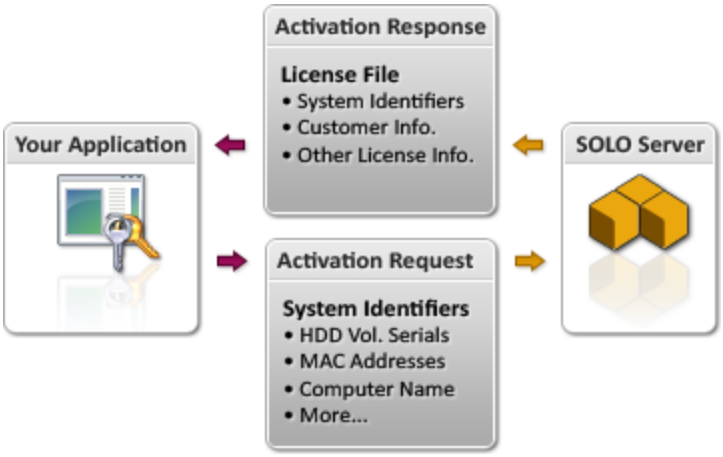
Any time a "back door" mechanism is put in place for "the good guys," that same mechanism is technically available to "the bad guys" as well. Consequently, it does not make sense to add any such mechanisms, as it would inevitably do more harm than good. The SoftwareKey System is not (and will not be) designed to include any such mechanisms. Furthermore, since we are not in the business of implementing cryptographic functions, we do feel it is very important for you to know which underlying libraries are used by Protection PLUS 5 SDK for cryptography.

The PLUSNative library uses OpenSSL for cryptographic functions. This is a widely used and trusted toolkit, which is also open source (meaning it's possible for you to review its source code).

The PLUSManaged library uses the Microsoft .NET Framework's built-in classes for cryptography, which often either has its own implementation, or uses Microsoft's CryptoAPI. If your .NET application is run using the Mono framework, then its own cryptographic functions are used in place of Microsoft's APIs (and Mono's implementation is also open source).

License Files

License files, which are typically issued by SOLO Server during activation, contain all of the license data or parameters needed by your application. This includes data such as the type of license being issued (i.e. time-limited, non-expiring, volume, etc...), expiration dates, what application features or modules may be used, what system is authorized to use the license, etc... This also contains additional data about the customer for whom the license is registered, information for obtaining technical support from your company, and any custom string or XML formatted data that you may optionally use to include additional, customized data in the license.



Format

Protection PLUS 5 SDK license files are formatted using Extensible Markup Language (XML). XML is a free and open standard which simply specifies how data may be encoded or formatted in a manner that does not limit the size or exact format. The primary benefits of using XML include: it is widely supported in nearly all industries, operating systems, and software development frameworks; it makes it possible to extend upon the license file format without breaking compatibility; and it allows you more flexibility and power to use highly customized data. The XML license files are also **encrypted and digitally signed** to prevent unauthorized access and manipulation. The actual content stored in the license file is defined by the **license file schema**.

Read-only versus Writable

Read-only license files are license files which must be encrypted and signed by SOLO Server. The advantage to these license files is that they are digitally signed with key data that is only partially known to your application. This means your application has enough of the key to confirm SOLO Server issued the file, but lacks the data needed to create these files. The benefit is that, since the key is not known to your application, it becomes difficult for hackers to compromise this key and write their own license files for your application. The drawback is that, since SOLO Server must issue these license files, Internet connectivity is required (either directly, or from another computer).

Some environments, including ones like remote locations (such as in a field or on the road), do not offer any Internet connectivity. Meanwhile, others are highly restrictive with Internet connectivity (often places such as hospitals, government and military offices, etc...). In these types of environments, it can be difficult or impractical to require Internet connectivity. Additionally, there might be other reasons why your application would need to be able to write to its license files freely as well (depending on your **licensing requirements**). In these situations, it is possible to use writable license files, which are license files that are encrypted and signed by your application (**read more about license files**). Although this means all key data needed to write license files is known to your application (which is less secure than using read-only licenses issued by SOLO Server), this affords you extra flexibility when needed.

Keep in mind that you need to make the best choices about what type of license file your application should use, and when it should use it. Our **sample applications** show you how you can use read-only licenses, writable licenses, and even a hybrid of both.

Storage

Out-of-the-box, Protection PLUS 5 SDK APIs support storing your application's license files on disk, in the Windows Registry, or in an image file. However, these same APIs also support loading and manipulating the files from memory, which affords you the flexibility of storing it where ever it suits your application. For example, you could store this in your application's database, embed it in other files, etc...

Additionally, keep in mind that permissions is always something you should consider when selecting your license file locations. Please read more about deploying applications protected with **PLUSManaged** and **PLUSNative** (as appropriate for your application) for more information.

In addition to file and registry storage, Protection PLUS 5 SDK supports storing and hiding the license file XML within image files using **steganography**.

Important

The steganography feature is presently a beta, and changes may be applied which could require minor source code modifications to be made in order to use a future release. Please keep in mind that **your testing** should be very thorough. Also, use your own images distributed with your application such as a splash screen image. Do not use an existing image such as the desktop wallpaper since another product could share that license location.

Consider the following when using the steganography feature:

- Currently only the Bitmap image format is supported. The image cannot use any compression.
- The Bitmap image must be a minimum of 50 Kilobytes in size. Larger images (100 KB or more) are recommended as your license data could grow and will fail to write if it is larger than the image.
- Use your own images, not system images.
- The image must be distributed with your application as Protection PLUS 5 SDK cannot create an image.
- On Windows OS, the access, modified, and created dates will not be updated on the image file when data is stored to it in order to help hide the write to the image.
- Keep a "clean" copy of the image. When testing, this allows you to copy the clean image over your test image to delete the internal license.

Aliases

Aliases are simply hidden copies of the license file. These should always be used with writable license files, as they help prevent users from doing things like back-dating their system and restoring old copies of license files. When using writable licenses files, these are especially important to use when issuing time-limited licenses (such as evaluation/trial or periodic/lease licenses), and when allowing transfer and deactivation of licenses.

Since these are simply copies of the license file, permissions is something you should also consider when selecting your alias locations. Please read more about deploying applications protected with**PLUSManaged** and **PLUSNative** (as appropriate for your application) for more information.

Alias files may also use the **steganography** feature for storage in a supported image format.

License File Schema

XML Formatting

Protection PLUS 5 SDK **license files** are simply XML documents that have been encrypted and digitally signed to **secure** the contents and prevent unauthorized modification. The encrypted license file is still an XML document. Once the license file has been decrypted and verified, you can read the various license fields to determine the state of the application's license. The XML schema, or general structure, of the license file is shown below. This schema differs from the encrypted license file, which is what you would normally see if you opened the license file in a text editor, to the decrypted license file, which is found in the application's memory. Several tables outline the many fields and provide a description of what these fields contain.

Encrypted License File

The following depicts the contents of an encrypted license file:

XML

```
<SoftwareKey>
   <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#" Id="PrivateData" Type="h-
   ttp://www.w3.org/2001/04/xmlenc#Element">
      <CipherData>
         <CipherValue>CEtwIAD3IhUUA1csp6IKqkwlzGoM78Ffq/SMs9lK3EayA2R9WPC3Pzk...</CipherValue>
      </CipherData>
   </EncryptedData>
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignatureValue>iehRwDe5eVNGZI+w/b/BnM7IxqK7MvK9t2t1zzyHwp+0gS06k...</SignatureValue>
   </Signature>
</SoftwareKey>
```

The `<CipherValue>` and `<SignatureValue>` node's content has been included to illustrate what the data might look like in an actual license file, but the data has been truncated.

Decrypted License File

The following depicts the contents of a decrypted license file. The links in comments, such as <!-- **License Fields** -->, are simply placeholders for the various nodes found in the corresponding tables below.

XML

```
<SoftwareKey>
   <PrivateData>
      <License>
         <!-- License Fields -->
         <ActivationData>
            <Identifiers>
               <SystemIdentifier name="..." type="..." value="..." />
            </Identifiers>
         </ActivationData>
         <Author>
            <!-- Author Fields -->
         </Author>
```

```
        <Customer>
          <!-- Customer Fields -->
        </Customer>
        <Distributor>
          <!-- Distributor Fields -->
        </Distributor>
        <Product>
          <!-- Product Fields -->
        </Product>
        <ProductOption>
          <!-- Product Option Fields -->
        </ProductOption>
      </License>
    </PrivateData>
</SoftwareKey>
```

The `<Identifiers>` node above may contain one or more `<SystemIdentifier>` child nodes. The `name`, `type` and `value` attributes would contain data based on the specific system identifier algorithm used, the number of identifiers present, and a unique hashed value. The "..." for the values are used to depict that this data is variable.

Field Listings

License Fields

In SOLO Server, all licenses are created from one Product Option record, and always belong to a customer.

Field	Description	XPath
ActivationData	Contains the system identifiers used to identify the system which is authorized to use the license file.	*/SoftwareKey/PrivateData/License/ActivationData*
ActivationPassword	A password, usually randomly generated by SOLO Server, which may be used with the corresponding License ID to activate or perform other licensing related functions.	*/SoftwareKey/PrivateData/License/ActivationPassword*
EffectiveEndDate	Corresponds with the "Download	*/SoftwareKey/PrivateData/License/EffectiveEndDate*

	Until" date on the SOLO Server license, and typically reflects the date in which the license expires (for time-limited licenses). However, it is also possible for your application to use this date for other purposes when the license is not time-limited, such as notifying customers when support/maintenance subscription periods are about to expire.	
EffectiveStartDate	The date in which the license is effective. This is typically the date in which the license was added in SOLO Server.	*/SoftwareKey/PrivateData/License/EffectiveStartDate*
ExternalReference1	An external reference number/string, typically generated by a third-party system or service.	/SoftwareKey/PrivateData/License/ExternalReference1
ExternalReference1Source	The name or description of the source of the ExternalReference1 value.	/SoftwareKey/PrivateData/License/ExternalReference1Source
ExternalReference2	An external reference number/string, typically generated by a third-party system or service.	/SoftwareKey/PrivateData/License/ExternalReference1
ExternalReference2Sour	The name or	/SoftwareKey/PrivateData/License/ExternalReference2So

ce	description of the source of the ExternalReference2 value.	urce
FormatVersion	The format version of the License File.	*/SoftwareKey/PrivateData/License/FormatVersion*
InstallationID	When an application is activated (typically using a License ID and password), an Installation ID is created so that **Electronic License Management (ELM)** may be leveraged on the individual system which was activated.	*/SoftwareKey/PrivateData/License/InstallationID*
InstallationName	An optional, human-readable description of an Installation ID (SOLO Server allows a maximum of 50 characters for this field). This is typically specified by the user to help describe which of several systems he or she owns is being activated (a user could enter something like "Home Desktop" or "Work Laptop").	*/SoftwareKey/PrivateData/License/InstallationName*
IsTestLicense	Identifies if the license is a test license signified by a value of 1. Test licenses are deleted from SOLO Server at the beginning of each month.	*/SoftwareKey/PrivateData/License/IsTestLicense*

LastUpdated *	Only available when using writable **license files**, this reflects the last date and time in which the application was run.	*/SoftwareKey/PrivateData/License/LastUpdated*
LatestVersion	The latest version of the product or application available (at the time the license file was issued).	*/SoftwareKey/PrivateData/License/LatestVersion*
LicenseCounter	An arbitrary, numeric value which may be used to track a number or count for a license requirement. This can be used for noting the number of feature or application uses allowed, the number of network seats allowed, etc..	*/SoftwareKey/PrivateData/License/LicenseCounter*
LicenseeEmail	The email address of the licensee, or the person for whom the license has been assigned.	*/SoftwareKey/PrivateData/License/LicenseeEmail*
LicenseeName	The name of the licensee, or the person for whom the license has been assigned.	*/SoftwareKey/PrivateData/License/LicenseeName*
LicenseCustomData	Custom string or XML formatted data specific to the license. All strings in the license file use UTF-8 encoding. Unless	*/SoftwareKey/PrivateData/License/LicenseCustomData*

	included in nested XML in this field, any Unicode characters in this field will be formatted with **XML escape sequences**.	
LicenseID	A unique, numeric identifier for the license issued (typically issued by SOLO Server).	*/SoftwareKey/PrivateData/License/LicenseID*
LicenseUpdate	A small, arbitrary string which may be used for sending customized license status updates.	*/SoftwareKey/PrivateData/License/LicenseUpdate*
ProductID	A unique, numeric identifier used to identify the Product for which a license was issued. This value is typically generated by SOLO Server.	*/SoftwareKey/PrivateData/License/ProductID*
ProdOptionID	A unique, numeric identifier used to identify the Product Option for which a license was issued. This value is typically generated by SOLO Server.	*/SoftwareKey/PrivateData/License/ProdOptionID*
QuantityOrdered	The quantity of the product ordered (usually 1). When using the E-Commerce features in SOLO Server, it is possible to configure your Product Option so that ordering	*/SoftwareKey/PrivateData/License/QuantityOrdered*

	more than one quantity generates a single license with extra activations. So for example, if you had a Product Option configured with 2 activations, you can configure the Product Option to create a single license with 6 activations when a customer orders a quantity of 3 (instead of creating 3 separate licenses, each with 2 activations).	
SerialNumber	The serial number assigned to the license, usually defined in SOLO Server.	*/SoftwareKey/PrivateData/License/SerialNumber*
SignatureDate	The date in which the license file itself was created and signed by SOLO Server.	*/SoftwareKey/PrivateData/License/SignatureDate*
TriggerCode	The Trigger Code number issued by SOLO Server. This is a numeric value between 1 and 50, and is typically used to reflect the type of license being issued (i.e. non-expiring, time-limited, etc...). This value is configured on the Product Option in SOLO Server.	*/SoftwareKey/PrivateData/License/TriggerCode*
TriggerCodeFixedValue	The Trigger Code Fixed Value issued	*/SoftwareKey/PrivateData/License/TriggerCodeFixedValue*

	by SOLO Server. This is a numeric value up to 14 bits in size (or a maximum value of 16383), which is typically used to conditionally enable application features, or distinguish between different product or application editions (i.e. "Express Edition" and "Enterprise Edition", etc...). This value is configured on the Product Option in SOLO Server.	
UserDefinedDate1 - UserDefinedDate5	User defined date fields 1 through 5, which may be used for arbitrary dates when needed for additional, customized license data.	*/SoftwareKey/PrivateData/License/UserDefinedDateN*
UserDefinedFloat1 - UserDefinedFloat5	User defined float fields 1 through 5, which may be used for arbitrary decimal values when needed for additional, customized license data.	*/SoftwareKey/PrivateData/License/UserDefinedFloatN*
UserDefinedNumber1 - UserDefinedNumber5	User defined number fields 1 through 5, which may be used for arbitrary integer values when needed for additional, customized license data.	*/SoftwareKey/PrivateData/License/UserDefinedNumberN*

UserDefinedString1 - UserDefinedString10	User defined string fields 1 through 10, which may be used for arbitrary alpha-numeric values when needed for additional, customized license data.	*/SoftwareKey/PrivateData/License/UserDefinedStringN*

* - The *LastUpdated* field is only present in self-signed writable license files.

Author Fields

You or your company is the "Author" of the software or application being licensed.

Field	Description
AuthorID	Numeric value which uniquely identifies your SOLO Server author account.
CompanyName	Your company name, as configured in your SOLO Server author account.
CustomData	Custom string or XML formatted data shared with all licenses created in your SOLO Server author account. All strings in the license file use UTF-8 encoding. Unless included in nested XML in this field, any Unicode characters in this field will be formatted with **XML escape sequences**.
SupportEmail	Your company's support email address, which your customers may use to obtain technical support.
SupportPhone	Your company's support phone number, which your customers may use to obtain technical support.
SupportSite	Your company's support web site, which your customers may use to obtain technical support and resources.

Customer Fields

The Customer represents your customer, for whom one or more licenses have been issued.

Field	Description
Address1	The customer's street address (line 1).
Address2	The customer's street address (line 2, optional).

City	The city where the customer is located.
CompanyName	The customer's company name.
Country	The country where the customer is located.
CustomerID	The unique SOLO Server Customer ID for this customer.
Email	The customer's email address.
ExcludeFromAll	Whether the customer is excluded from all email marketing.
Fax	The customer's fax number.
FirstName	The customer's first name.
LastName	The customer's last name.
NotifyPartners	Whether the customer will receive email alerts from trusted partners.
NotifyProducts	Whether the customer will receive email alerts for products updates.
Phone	The customer's phone number.
PostalCode	The postal code where the customer is located.
StateProvince	The state or province where the customer is located.
Unregistered	Whether or not the customer is unregistered.

Distributor Fields

In SOLO Server, a distributor represents a company or individual that either resells your products or applications, or an affiliate which refers customers and prospects to your web site. The distributor information can be particularly useful in scenarios when the distributor acts like a reseller, and also provides its customers with technical support.

Field	Description
Address1	The distributor's street address (line 1).
Address2	The distributor's street address (line 2).

Address3	The distributor's street address (line 3).
CompanyName	The distributor's company name.
Country	The country where the distributor is located.
DistributorID	The distributor's unique ID.
Email	The distributor's email address.
Phone	The distributor's phone number.
Type	The type of distributor.
WebSite	The distributor's web site.

Product Fields

Each product defined typically represents the product or application being licensed (i.e. "XYZ Product"). Products must contain one or more Product Options, as all licenses are created from Product Options.

Field	Description
CustomData	Custom string or XML formatted data shared by all licenses which are created with the any of the Product Options which belong to the Product. All strings in the license file use UTF-8 encoding. Unless included in nested XML in this field, any Unicode characters in this field will be formatted with **XML escape sequences**.
ProductName	The name of the Product or application.

Product Option Fields

Product Options define licensing or purchasing options available under a product. These can reflect any number of unique licensing and/or purchasing options, for example: "1 Year Subscription", "1 Year Subscription with Backup CD", or "1 Year Subscription Renewal".

Field	Description
CustomData	Custom string or XML formatted data shared by all licenses which are created with the Product Option. All strings in the license file use UTF-8 encoding. Unless included in nested XML in this field, any Unicode characters in this field will be formatted with **XML escape sequences**.

OptionName	The name of the Product Option, which typically reflects the varying licensing or purchasing options available for your application/product.
OptionType	The Product Option Type, which describes the type of license being issued (i.e. a license which uses standard activation, volume license, etc...)

System Identification

System identification is where your application obtains some information about the system on which it is running. Typically, any license which requires activation uses information about the system so that the specific system may be authorized. The Protection PLUS 5 SDK APIs assist you with authorizing and validating this information by making several "System Identifier Algorithms" and supporting functions/methods available to you. These algorithms can be used to generate "System Identifiers" using any combination of information including (but not limited to) the different identifiers listed in these topics:

- **PLUSManaged API System Identification Algorithms**
- **PLUSNative API System Identification Algorithms**

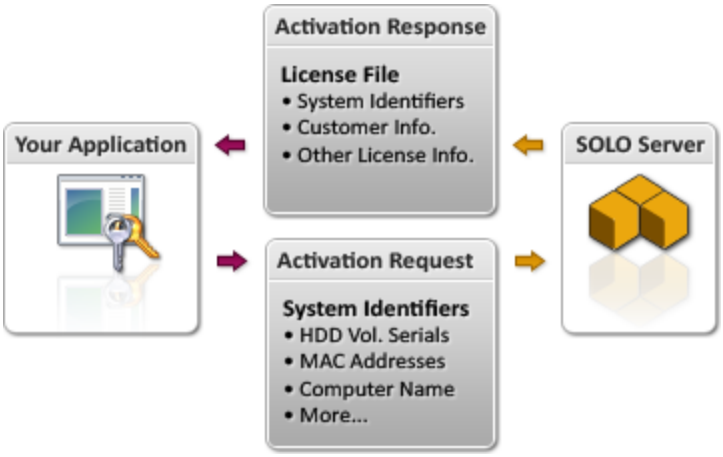
Furthermore, the Protection PLUS 5 SDK APIs are so flexible and robust, they allow you to specify your own types of identifiers.

Important

It is imperative to keep in mind that you are responsible for testing any custom system identifier algorithms you create to ensure that the reliability and uniqueness of the identifiers meet your expectations and requirements.

System Authorization

A system is authorized for a license during activation with SOLO Server. This allows SOLO Server to include the information about the system in the license file issued as illustrated below.



To further explain the illustration, here is the full process it depicts:

1. A user runs your application, and decides to activate. At this point, your application uses the Protection PLUS 5 SDK APIs to generate an activation request.
2. While generating the activation request, any information you opt to use to identify the current system (on which activation is being requested) is gathered and included in the activation request.
3. The activation request is sent to SOLO Server, which validates the request, and compiles an activation response.
4. Part of the activation response generated by SOLO Server is the license file, which will also include the system identifiers provided in the activation request. This is how that system is authorized to use that specific license file.
5. Your application receives the activation response, and saves the license file it received from SOLO Server. Your application may then evaluate the data in the license file to ensure the system running your application is the system which was authorized in the license file.

Allowing Acceptable Change

It is not uncommon for a customer to change some of his or her computer's hardware after activating your software. Changes can occur when components and drivers fail, and it is often (if not generally) ideal for your application to allow for some amount of change without requiring activation. The Protection PLUS 5 SDK APIs give you access to the data necessary to compare and contrast information about the system currently running versus the system which was authorized at the time of activation. So for example, if you used MAC addresses from the system's Network Interface Cards (NICs), you might expect the number of NICs present to change on mobile computers or laptops. This is because many laptops might have removable hardware connected and disconnected regularly, which sometimes includes docking stations. With this in mind, your application can leverage the Protection PLUS 5 SDK APIs to ensure at least a certain number of NICs match, as it best suits your application's licensing requirements. Of course, this principle can be applied with any type and combination of system identifiers.

Privacy

Using a large amount of information about a system to authorize it has the benefit of allowing for more flexible matching fuzzy-logic (to allow some acceptable amount of change to occur without requiring a new activation) and allows you to better prevent other, very similar systems from being inadvertently authorized to use the same license file. However, collecting this kind of information could cause privacy concerns for your users. The system identification logic in Protection PLUS 5 SDK and SOLO Server relies on using hashes of information (**more about hashes**). Using the one-way cryptographic hashes prevents anyone and everyone from using data showing what hardware your customers use for anything other than the intended purpose (system authorization). The Protection PLUS 5 SDK APIs are not designed to enable you to use this data for purposes other than system identification and authorization for licensing purposes, and SOLO Server is also not designed to make this data useful or available outside of the intended capacity.

Electronic License Management (ELM)

Electronic License Management (ELM) is a core feature of **SOLO Server**, which allows you to maintain control of your software after it leaves your possession. It also allows you to automate the disbursement of software updates while optionally collecting usage information and verifying that a customer's license is still valid.

If you choose not to use SOLO Server, Protection PLUS 5 SDK includes **License Manager**. This allows you to activate your customers manually but doesn't provide the other ELM features in this topic.

Activation

The heart of the software licensing and copy protection is product activation. Every copy of your software must be activated when installed on a new computer. In addition, creating, maintaining, and distributing different builds of your application, where each build fulfills a unique set of licensing requirements, can result in a significant amount of unnecessary effort. Distributing separate builds of your application (each making different sets of features and modules available based on what license was purchased) is less than ideal, as it would be just as possible for anyone to arbitrarily distribute such packages. Instead, activation via Protection PLUS 5 SDK allows a single build of your application to change the state of the license for a computer, network, and/or tailored licensing requirements.

In addition to automating the activation process, SOLO Server also allows you to keep track of activations, allow reactivations on the same computer, etc. These features are not available with License Manager.

Background Checking & Refreshing

Imagine you are using SOLO Server's e-commerce features, you sell a license for your application, the customer activates it, and a few weeks later you find the license was purchased with a stolen or lost credit card. Or another scenario could be that a customer has called and activated your application, and is requesting a new activation claiming his or her original computer is broken and had to be replaced. Protection PLUS 5 SDK APIs allow you to automatically check with SOLO Server periodically to verify the status of the license. Should you encounter situations like the ones described here, you can disable an individual system previously activated or the entire license (and all systems activated with it), and rest easy that the system revoked will soon no longer be licensed and able to run.

Additionally, some applications may require time-limited licenses which are activated (known as periodic or lease licenses). SOLO Server's rich e-commerce features can help you automate accepting payments (even automatic, recurring payment) while making it possible for your software to automatically update its expiration date. So for example, let's say your customers may only use the licensed application or service for 30 days before payment is necessary again. Once the next payment is processed after 30 days, your application can automatically refresh its license without requiring customers to go through the process of activating again, which will result in your application responding to the new expiration date automatically. This level of automation gives you and your users the simplest and most convenient experience possible.

Furthermore, any information in your application's license files can be edited through SOLO Server's web interface and pushed down to your customer's system(s) by simply refreshing the license. This includes customer data, product data, information about how to reach your company for technical support, customized license parameters, etc..

Deactivation & Transfer

There are three ways a license may be deactivated. The first is where you, the author of the application, deactivates the license remotely through SOLO Server. In this scenario, your application needs to rely on the background checking and refreshing features of Protection PLUS 5 SDK and SOLO Server to detect and respond to this event appropriately. The second is similar, but allows the customer to **deactivate the license remotely though the customer license portal**.

The third type of deactivation is one which is initiated by someone using your application. This is a very simple and convenient way for your customers to essentially migrate (or transfer) the license from one system to another. An example of when a customer would want to do this would be when he or she is upgrading from one computer to another. Allowing your customers to do this online in this manner is a convenience to them, and reduces your support overhead.

Important

It is very important to rely on background checking and refreshing when allowing users to deactivate your application! Automated background checking helps prevent users from restoring the entire system/computer from an image or snapshot taken before deactivation occurred.

Using Trigger Codes for Telephone Activations

Trigger code validation is a means of achieving software activation through the manual exchange of numeric values (in other words, it is a challenge-response mechanism). Allowing activation through manually exchanged numeric codes is convenient when you need to be able to activate remote systems which lack Internet connectivity, as it is easy to exchange these codes via a telephone call, text messages, etc... Trigger Codes are only capable of carrying a very limited, numeric payload, but this limitation is easily overcome by coupling trigger code validation with the **downloadable license file** (which can contain any and all license data).

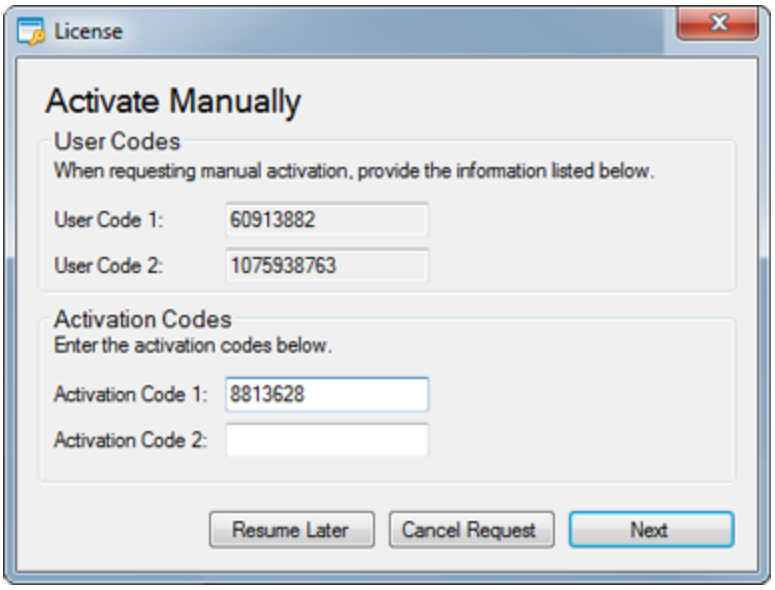
Important

Keep in mind that any challenge-response mechanisms such as trigger code validation can be reverse engineered (similar to how "key generators" are available on the Internet for many popular software applications). Therefore, it is best to use Protection PLUS 5 SDK's standard online and manual activation features when possible, which offer much stronger **security** than trigger codes.

Overview of Trigger Codes

The Basics

Below is an example of the kind of screen your users will see when activating manually using trigger codes.



The trigger code activation process is as follows:

1. The protected application shows a dialog similar to the one shown above. In some cases, you may wish to ask your users to enter a License ID and password, which could be stored and used for online validation at a later time.
2. The user contacts you or your company, and provides the User Code 1 and User Code 2 values displayed on the activation dialog.
3. You (or your company) generate and respond to the user with the Activation Code value(s).
4. The protected application validates the Activation Code(s), and alters its license as appropriate.

The Bits and Pieces

A summary of the different pieces of information used in trigger code processing is below. This will help you establish a more complete understanding of how trigger codes work.

Important

When using activation with SOLO Server, the Trigger Code Seed and RegKey2 Seed values in your application's source code must match in all relevant product options configured in SOLO Server (which may be used to create License IDs used to activate your application over the Internet).

Trigger Code Seed

This is the "seed" value used when generating *Activation Code 1*. Typically, this value should be unique between each of your products/applications so that you may prevent them from being cross-activated. (In other words, using a unique *Trigger Code Seed* value for each product prevents activation codes generated for "Product A" from being used to activate "Product B"). The value specified must be a number between 1 and 65535. When allowing activation through SOLO Server, your application's source code and the corresponding product option(s) in SOLO Server must have matching values.

RegKey2 Seed

This is the "seed" value used when generating *Activation Code 2*. Typically, this value is also unique between each of your products/applications so that you may prevent them from being cross-activated. The value specified must be a number between 1 and 255. When allowing activation through SOLO Server, your application's source code and the corresponding product option(s) in SOLO Server must have matching values.

Trigger Code Number

A Trigger Code Number is simply a number between 1 and 50, which defines the action(s) which will be taken when *Activation Code 1* is successfully validated on your customer's computer. When a customer requests activation, you or your company is responsible for selecting the appropriate trigger code number when generating the activation code(s). (When using SOLO Server, this is configured on the product option selected when the license was created or purchased.) As the application developer, you are free to assign and program your own set of actions for any given *Trigger Code Number*. So for example, you could program your application to activate a non-expiring (or perpetual) license for your application when it receives a trigger code number of 1. At the same time, your application could also be programmed to activate a time-limited (or a "lease" or "periodic") license for some arbitrary number of days when it receives a trigger code number of 10.

User Code 1

Also known as the "session code" to former Protection PLUS 4 users, this value is a randomized value that makes activation codes unique to each activation attempt. Its purpose is to prevent end-users from using an activation code more than once, which is very important when an activation code does something such as incrementing the number of network seats allowed. When users contact your company to activate, they will be required to provide this value.

User Code 2

Also known as the "Computer ID" to former Protection PLUS 4 users, this value is generated from device-specific information. Its purpose is to make each activation code unique to the PC for which it was issued, and prevent users from activating "Computer B" with an activation code generated for "Computer A". When a users contact your company to activate, they will be required to provide this value.

Activation Code 1

When a user requests activation, this value is generated by you or your company. This value must always be provided to your end-users to complete an activation. Once your protected application has successfully validated Activation Code 1, it receives the Trigger Code Number, which allows your protected application to alter its license file as appropriate.

Activation Code 2

When a user requests activation, this value may (optionally) be generated by you or your company in order to send a secondary, numeric value to your application during activation. The decoded value sent to your application through *Activation Code 2* may only be up to 14 bits in length (possible values range from 0 to 16383). For example, when issuing activations for a time-limited (or a "lease" or "periodic") license, this value can include the number of days which the protected application is allowed to run.

Generating Activation Codes

In most cases, SOLO Server is used to generate activations codes. This involves **creating a new license**, and clicking the *Manual Activation* link at the top of the *License Details* page. From here, you can enter the *user code* values given to you by your end-user, and generate the *activation codes*. SOLO Server keeps a log of the activation history for all types of activations (online, manual, and trigger code activations), which affords you many benefits of **Electronic License Management** regarding activation and background checking. However, it is important to note that features such as license refreshing and deactivation may not be available when using trigger code activation alone. (**Downloadable licenses** are an exception to this limitation.)

Protection PLUS 5 SDK does not presently include a means to generate activation codes for trigger code activation. However, if you already have a Protection PLUS 4 license, you can continue to use LFEdit for this purpose. If you have a need to generate these activation codes without SOLO Server, and you do not have Protection PLUS 4, please **contact us**.

Licensing Overview

The primary goal of adding licensing logic to your software is to ensure that anyone using your application has access to what he or she is entitled to, for as long as he or she is entitled to it - nothing more, and nothing less. This topic summarizes some of the many different ways of managing access to your application.

Common Restrictions

You can choose to implement any combination of restrictions to limit the use of your application. The most common restrictions are described below, but keep in mind that Protection PLUS 5 SDK is very flexible, and does not limit you to the restrictions explained here.

Time

The most common restriction is limited time. This is where licensing restricts the amount of time the application may be used before a purchase is required. As an example, this could simply be a free evaluation which only runs for 30 days. With time-limited licenses, your application would use the `EffectiveEndDate` **in the license file** to enforce the expiration date.

Uses

Another common restriction is to limit the amount of times a certain feature can be performed or the amount of times the application itself may be run. This allows use for a given number of times before requiring a purchase to increase the counter value. The counter value is an arbitrary numeric value "passed down" (or supplied) to the licensed application during the software activation process. For these scenarios, a writable **license file** is often necessary, as it allows your application to use the `LicenseCounter` field (recommended) or any of the numeric, user-defined properties to keep track of uses/counters. When using the `LicenseCounter` field, the customer can automatically purchase additional uses through SOLO Server and then refresh their license. For a more secure implementation, consider storing the usage counter in SOLO Server to track pay-per-use applications more securely (Internet connection always required).

Features

Limitations on application features can also be implemented in a variety of ways.

First, applications that contain features which should be evaluated may want to make sure the feature is only used a reasonable number of times without allowing indefinite use. For these scenarios, a writable **license file** is often required, as it allows the application to use any of the numeric user-defined properties to keep track of uses/-counters. An example of this scenario could include an application that transfers videos to a portable device, where the application might only permit the user to transfer a handful of such files during evaluation (so users feel comfortable it works as expected before purchasing). After the user purchases and **activates** the application, he or she can then be allowed to transfer any number of videos.

In other cases, applications might have features that could alter output in a way that would require purchase for practical use. For example, if the application produces some kind of document output (such as a PDF document), it could overlay a word or phrase (like "DRAFT COPY" or "EVALUATION VERSION") on the document to prevent practical use of the application's output until a license has been purchased and activated.

Other applications may simply have features completely disabled during the evaluation. An example could include image manipulation software which shows a limited preview of its output, but does not allow the manipulation/transformation to be applied to the photo and saved during the evaluation.

Furthermore, it is sometimes best to license additional features as if they are each their own application. One example of where this may be appropriate is an application which is licensed itself, but also supports dynamically load-

ing additional plug-ins at runtime. The additional plug-ins which add additional features and functionality to the application could then be evaluated, purchased, and activated independently from the application's license.

Common License Models

A license model simply describes any type of license you may wish to support in your application, which typically also provides an abbreviated way to describe the type of restrictions enforced.

Evaluation

Evaluation (also commonly referred to as trial, demo, or shareware) licenses are licenses issued to prospective customers. Evaluations provide an opportunity to set a good impression with your prospects, which often compels them to make a purchase to continue using your application. Additionally, an evaluation by a prospect could be a collaborative effort, so this can also afford your prospects the freedom to share the software amongst colleagues freely. This enables you to increase exposure of your application, which can lead to an increase in sales. Protection PLUS 5 SDK offers two primary means of allowing prospects to evaluate your software.

Self-Issued Evaluations

Although it is possible to implement evaluations of your application that require the customer to activate before it starts, this is not a requirement of Protection PLUS 5 SDK. Using a writable **license file** with aliases makes it difficult for users to back-date their computer to gain additional time, and also remain on the computer when your application is uninstalled so that re-installing your application does not afford the user a new trial period. Relying on aliases in your application can be preferential for many reasons, including:

- As a software publisher, you want your prospective customers to be able to easily access your product evaluations without the possibility of being hindered or "put-off" by an activation process.
- Issuing License IDs for activation of your evaluations comes with the risk that the prospective customer might not receive or might lose track of this information (which would be required for activation). Without knowing where to go to get this information, the prospect might disregard evaluation of your application entirely, which means a potentially lost sale.
- A happy or enthusiastic prospective customer may send a copy of your application to a friend. Ideally, the prospect's friend should be able to experience your application's evaluation very easily, which could result in additional sales and impulse purchases.
- Software publishers that use SOLO Server's evaluation/trial tracking services can present a form requesting basic customer contact and demographic information. Evaluation requests are stored in a separate database table, can be reported on or exported, and marketing information can automatically be sent to the evaluator. The link to download the application can be sent to the email address provided on the form. Failure to enter at least a valid email address will result in the prospect not receiving the link to download. Unfortunately, not all potential customers will use their real information on this form. Requiring the customer to activate their evaluation means that the evaluation request information will need to go into the SOLO Server Customer table, instead of this separate table. This means that any fictitious (and sometimes profane) information entered on the form will be present in your real customer database. Some software publishers who have implemented an activation procedure for evaluations have provided feedback that they regret filling their SOLO Server Customer database with this information.
- With the proliferation of free e-mail services (Yahoo! mail, Gmail, Hotmail, etc.), a new e-mail address can be created within a matter of minutes. If your evaluations require activation and you limit evaluation requests to unique email addresses, someone with the evaluation can easily sign-up for a new, free e-mail address, request a License ID for the evaluation, then keep extending the same evaluation on the same PC (unless you put special logic in the application to only allow an evaluation/trial period that cannot be extended). This potential weakness can be avoided by allowing your application to automatically establish an evaluation period for your prospects, as it would make it practical to require prospects to contact you for any evaluation extensions (which gives you the opportunity to verify the person's identity, and obtain additional sales lead information to follow-up with later).

Keep in mind that the above reasons might make it look appealing to implement the recommended evaluation functionality, it is only reasonable to also mention potential drawbacks:

- Self-issued evaluations rely on aliases (or hidden copies of the application's license file). If a hacker manages to find and delete all aliases and the license file, the current computer would appear to your application as though it had never run your application previously. This would result in your application automatically issuing a new evaluation period automatically.
- Although writable license files are secured, they are slightly less secure than read-only license files by nature. The sample applications show you how to use writable license files, the additional validation required for writable license files. Further more, certain **sample applications** illustrate how your application can use a writable license file for evaluations and a read-only license file for activated licenses.

You can implement writable license files with the **PLUSManaged API** or the **PLUSNative API** to enable your application to issue evaluation periods automatically.

Using SOLO Server to Issue Evaluations

There are some cases where it may not be acceptable for just anyone to install the application and automatically receive an evaluation. This could, for example, include cases where purchase and/or activation is required to evaluate the application. Consequently, requiring activation and a higher level of security through the use of read-only **license files** is often desirable, if not required, in these types of scenarios.

Protection PLUS 5 SDK gives you the flexibility of defining your own license models and their respective behaviors by using the various properties available to you in the license file. Our recommendation is to **configure your product option** in SOLO Server, and then edit it as follows:

- Pick a number between 1 and 50, and simply assign that number to a type of license. When configuring the evaluation/trial product option in SOLO Server, set the *Trigger Code #* field to this value. The `TriggerCode` **in the license file** will contain the value from the product option when the license file is issued by SOLO Server.
- If you have additional numeric data relevant to the type of license being issued, the value in the *TC Fixed Value* field in the Product Option configuration will be available in `TriggerCodeFixedValue` when the license file is issued by SOLO Server.
- Also, when configuring the Product Option in SOLO Server, the *Days to D/L* field may also be used to define the number of days for which an issued license should be effective/valid. When a new License ID is created for this Product Option in SOLO Server, its expiration date will be set to the specified number of days in the future from the date in which it was created. So for example, if you create a 30 day license on June 1 , it will be set to expire on July 1. The expiration date of any given license will be in `EffectiveEndDate`. You are also able to edit the expiration date in SOLO Server authors management interface, and have your application refresh its license file to retrieve the new date.

Important

When configuring your product option, Protection PLUS 5 SDK requires the *Issue Installation ID* option to be checked!

Once you have defined and configured the product option, you can then expand on the license validation in your implementation as necessary. For example, if you picked a *Trigger Code #* of 1 to represent evaluation licenses, you may always want to verify that the date in `EffectiveEndDate` is not a past day when the value of `TriggerCode` is 1.

Periodic

Periodic licenses are time-limited licenses (similar to evaluations) which always require activation to be used. This type of license model is suitable for scenarios where users are required to periodically pay to renew license to use the application.

Non-expiring

Non-expiring licenses (also known as perpetual licenses) are licenses which, once activated, never expire. While this means that the license to use the software never expires, it is also common to use non-expiring licenses in software while using SOLO Server to maintain a subscription that affords your customers technical support and version updates.

Advanced Models

There are wide variety of license models that you could potentially implement using Protection PLUS 5 SDK APIs. This includes, but is not limited to:

- **Network Floating Licenses**
- **Volume and Downloadable Licenses**

Network Floating Licensing

Network Floating Licensing is where an application may be restricted to running on a specific network, and restricted to a certain number of concurrent seats (where a seat may be a user or running instance of your application). Read our blog post for a general overview as well as two approaches to network floating licensing: **Controlling Concurrent User Access with Network Floating Licensing**.

Semaphore Files

A Network Semaphore File is a file, which contains no useful data, stored on a common network directory used to assist in enforcing a Network Floating License in an application. One file is created and locked exclusively for each active workstation or instance of your protected application. Other LAN users attempting to run the application will not be able to use any semaphore files that are locked, which is how this can enforce license compliance which limits the number of network "seats." This approach is ideal for many, as there is no need to manage a separate server or appliance to enforce restrictions. This feature is only supported when both the client and server use Windows Vista or Windows Server 2008 and later.

Important

The computer hosting the Windows (SMB/CIFS) share (the server) which houses the semaphore files is ultimately responsible for handling when a file is unlocked. In most cases, applications which are terminated gracefully will unlock and delete their semaphore files. If the application crashes, however, the server will eventually unlock the semaphore file, but it will not delete the file. Having many of these files present can lead to some performance degradation, as each of these orphaned files can take a while to try to lock or delete. For this reason, you have the option to establish a "clean-up thread" (the `NetworkSemaphore` constructor in PLUSManaged has a `runCleanupThread` argument, and PLUSNative has a `SK_PLUS_NetworkSemaphoreCleanup` function that can do this for you), which establishes active processes that try to delete orphaned files quietly, in the background. Of course, this will increase the amount of bandwidth used, but can help increase performance keeping the share location clear of any orphaned files.

This functionality is supported on environments where semaphore files are hosted on a Windows (SMB/CIFS) share, which functions best on a Local Area Network (LAN). While this may be used in a Wide Area Network (WAN) environment (which includes scenarios where users access a given "LAN" over a Virtual Private Network [VPN] connection), it is strongly advised that testing be done before deploying to any given WAN environment. This is because the performance of this feature is centered around the Windows (SMB/CIFS) shares being used to host the semaphore files, and WAN/VPN environments can see much slower performance than a typical LAN configuration.

Additionally, any users who will use an application using this type of licensing will need access to read, create, write, modify, and delete the semaphore files.

A good starting point for seeing how this kind of functionality works and can be integrated into your application is to test and review the **sample applications**.

Cloud-Controlled with SOLO Server

In cases where it is not practical or feasible to use semaphore files (such as for high-load and/or WAN environments), it is also possible to use SOLO Server to enforce concurrent seat limitations. In this case, SOLO Server web services are used to manage the state of each "session." These web services may also be used to automate authorization of "check-outs" for temporary, off-line use of the licensed application. You can read the **overview on Cloud-Controlled Floating Network Licensing** using SOLO Server for additional details.

Working with Network Floating Licenses in SOLO Server

After **adding a license in SOLO Server**, you can edit the license details to customize additional settings. When editing these details, you can change the number of network seats a given license may allocate by modifying the *License Counter* value. When using semaphores for network floating, the license will need to be refreshed after applying this change.

Cloud-Controlled Floating Network Licensing using SOLO Server

In cases where it is not practical or feasible to use semaphore files (such as for high-load and/or WAN environments), it is also possible to use SOLO Server to enforce concurrent seat limitations. In this case, SOLO Server web services are used to manage the state of each "session." These web services may also be used to automate authorization of "check-outs" for temporary, off-line use of the licensed application. If you would like to use this functionality or see it demonstrated, **contact us**.

Please read our **Controlling Concurrent User Access with Network Floating Licensing** blog post for great high level information.

Understanding Cloud-Controlled Floating Network Licensing using SOLO Server

Key Terms

To better understand Cloud-Controlled Floating Network Licensing using SOLO Server, you first need to understand several key terms:

Session

> A network session represents a seat that is in use, and one session is typically created for each instance of the application being run.

Session ID

> An alphanumeric value used to uniquely identify a given session.

Seat

> A seat is simply a network session which may be created. In this sense, if you were to allow a maximum of 10 concurrent network sessions for a given license, you might rephrase that as a "10-seat network license." In this example, when 10 seats in total are allowed, and 7 sessions are active, only 3 seats would still be available (corresponding with the ability to create only 3 additional sessions).

Orphaned Session

> This is a session which has not been closed, but has expired and is no longer valid. This could be the result of not performing a session close action, or from the licensed application terminating abnormally.

UTC

> UTC stands for **U**niversal **T**ime, **C**oordinated. With UTC, the date and time is always the same regardless of what time-zone you are physically located in, or whether or not any kind of daylight saving time is in effect in your time-zone. SOLO Server and the Protection PLUS 5 SDK APIs use UTC for all of the dates and times used for managing network sessions; however, the sample applications will translate and display the Allocated Until Date to you in your local system time. In layman terms, time in UTC is the same as Greenwich Mean Time (GMT). More information about UTC is available at **http://en.wikipedia.org/wiki/UTC**.

Allocated Date

> The date and time (in UTC) in which the network session was opened.

Allocated Until Date

> The date and time (in UTC) in which a network session is no longer valid. This date is extended when the licensed application performs a poll. The date is also determined by the configurable parameters defined

later, and is calculated by SOLO Server as follows:
*Allocated Until Date = (Current Date and Time) + ((Poll Frequency) + (Poll Retry Count * Poll Retry Frequency)).*
When a session is checked out for a given duration, the Allocated Until Date is then set to the (Current Date and Time) + (the specified duration).

Session Certificate File

When a network session has been checked-out for offline use, the information returned by SOLO Server comes back in a format that is designed to be used as a certificate file. This file may then be saved to the local computer's file system so it may be used later to restore the checked-out session.

Configurable Parameters

When using Cloud-Controlled Floating Network Licensing using SOLO Server, there are a variety of parameters that can be configured in SOLO Server to control how protected applications behave. The benefit of configuring these options in SOLO Server is that it enables you to make adjustments on the fly, as your applications can pull the new settings down the next time they open or poll a session. This means you can respond to time limits being too restricted or to relaxed, or respond to heavy load by adjusting poll and retry frequency, etc...

Poll Frequency

This is the frequency, in seconds, in which the licensed application should poll against SOLO Server.

Poll Retry Count

If a poll fails for any reason, this is the number of times the licensed application should try polling again.

Poll Retry Frequency

If a poll fails for any reason, this is the amount of time that should pass, in seconds, between attempts to retry the poll.

Allow Temporary Overages

If enabled, this will allow temporary overages to occur within the set maximum overage period. An example scenario of where this could be useful could include a user shutting the lid on a laptop before heading out to lunch, which causes the computer to suspend or hibernate. In this state, the computer is unable to perform any polls to keep the session active. When the user lifts the lid on the laptop and returns to its normal, running state, the session would no longer be active, and this could result in the user being locked out of the application if other users have since consumed all available seats. By allowing temporary overages, this will allow the user to resume a session which has since expired within the allowed overage period; however, this would result in the possibility of more seats being used than licensed temporarily. No new sessions may be open until enough active sessions have closed or expired to fall back under the seats allowed.

Maximum Overage Period

Please refer to the description of Allow Temporary Overages above. This is the time, in seconds, in which a session is allowed to be expired before polling again to resume its session, even if it means an overage will occur.

Checkout Duration Minimum

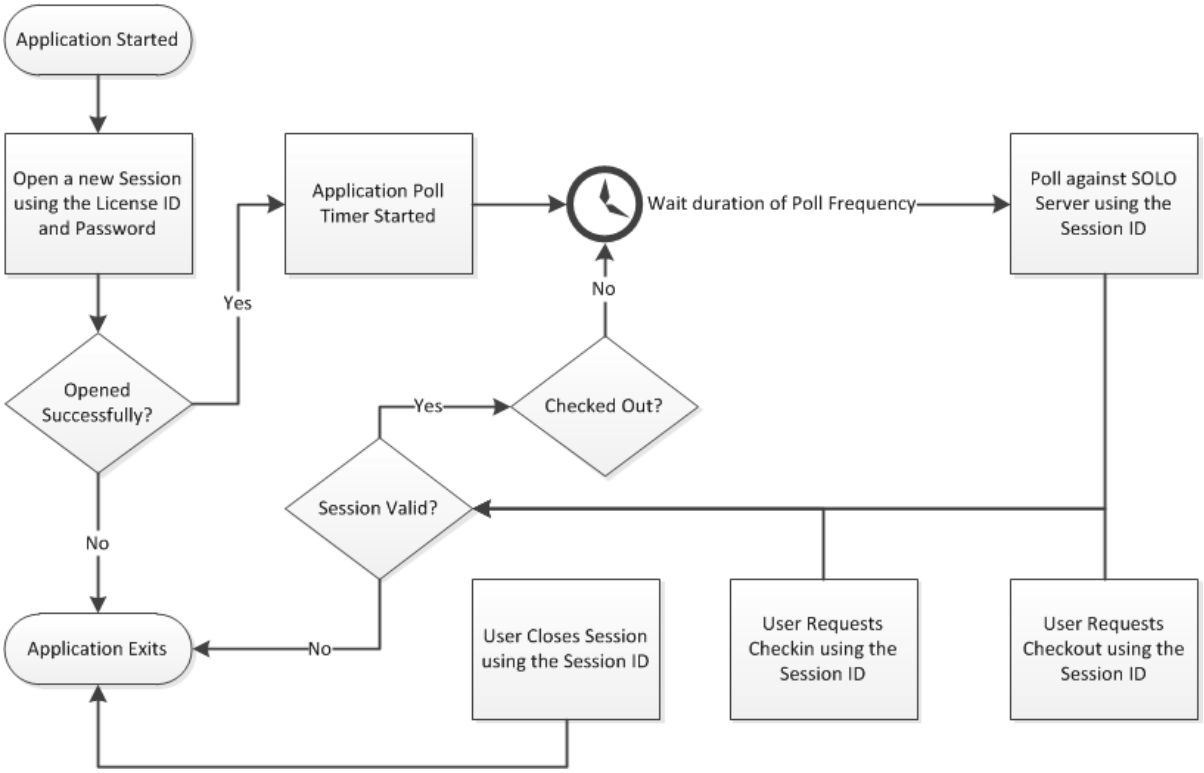
The minimum amount of time, in hours, which may be requested for a session checkout request.

Checkout Duration Maximum

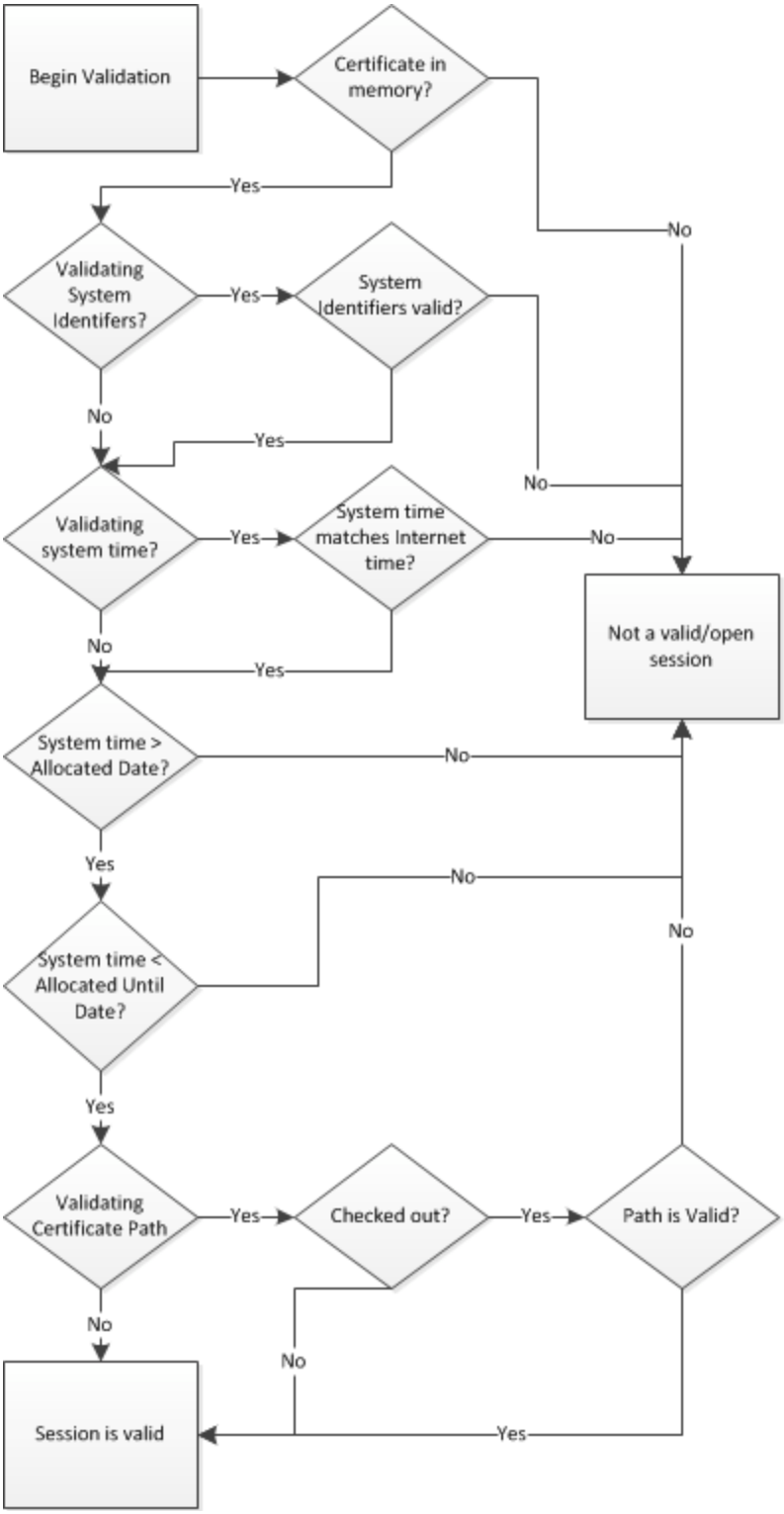
The maximum amount of time, in hours, which may be requested for a session checkout request.

How it Works

Applications that use Cloud-Controlled Floating Network Licensing using SOLO Server will perform a series of actions in order to create and manage concurrent sessions. A typical application work-flow is shown below:



As you can see in the flow chart above, session validation is meant to occur after every type of action occurs, so this is a very important piece to consider. While the Protection PLUS 5 SDK APIs simplify validation, below is an illustration of how the validation works.

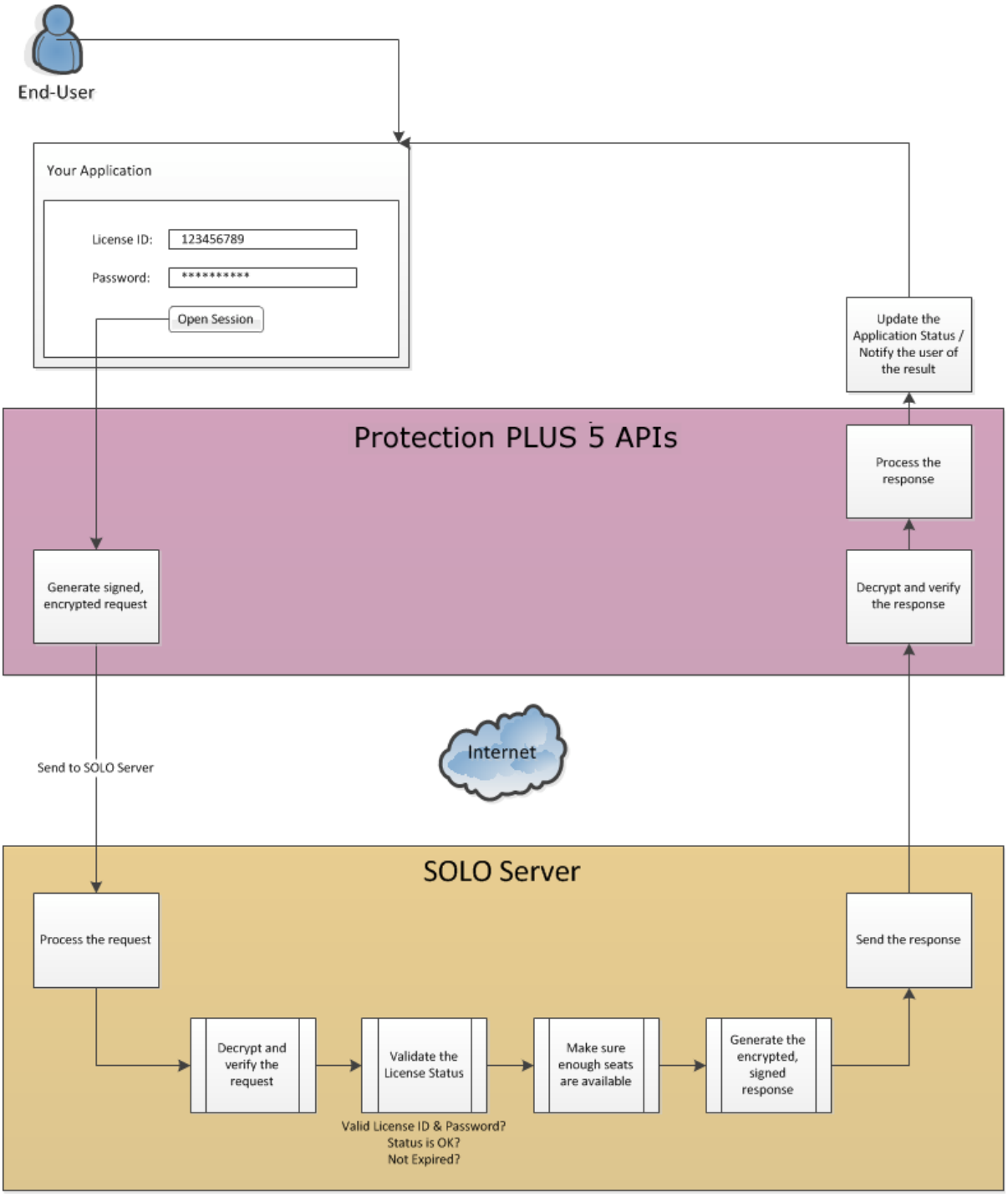

Begin Validation
Certificate in memory?
Yes
No
Validating System Identifers?
Yes
System Identifiers valid?
No
Yes
No
Validating system time?
Yes
System time matches Internet time?
No
No
Yes
Not a valid/open session
System time > Allocated Date?
No
Yes
No
System time < Allocated Until Date?
No
Yes
Validating Certificate Path
Yes
Checked out?
Yes
Path is Valid?
No
No
Session is valid
Yes

Actions

As described in the section immediately above, a variety of actions are performed to create and manage a concurrent network session. The actions that are supported by the Protection PLUS 5 SDK APIs are described and illustrated in detail below.

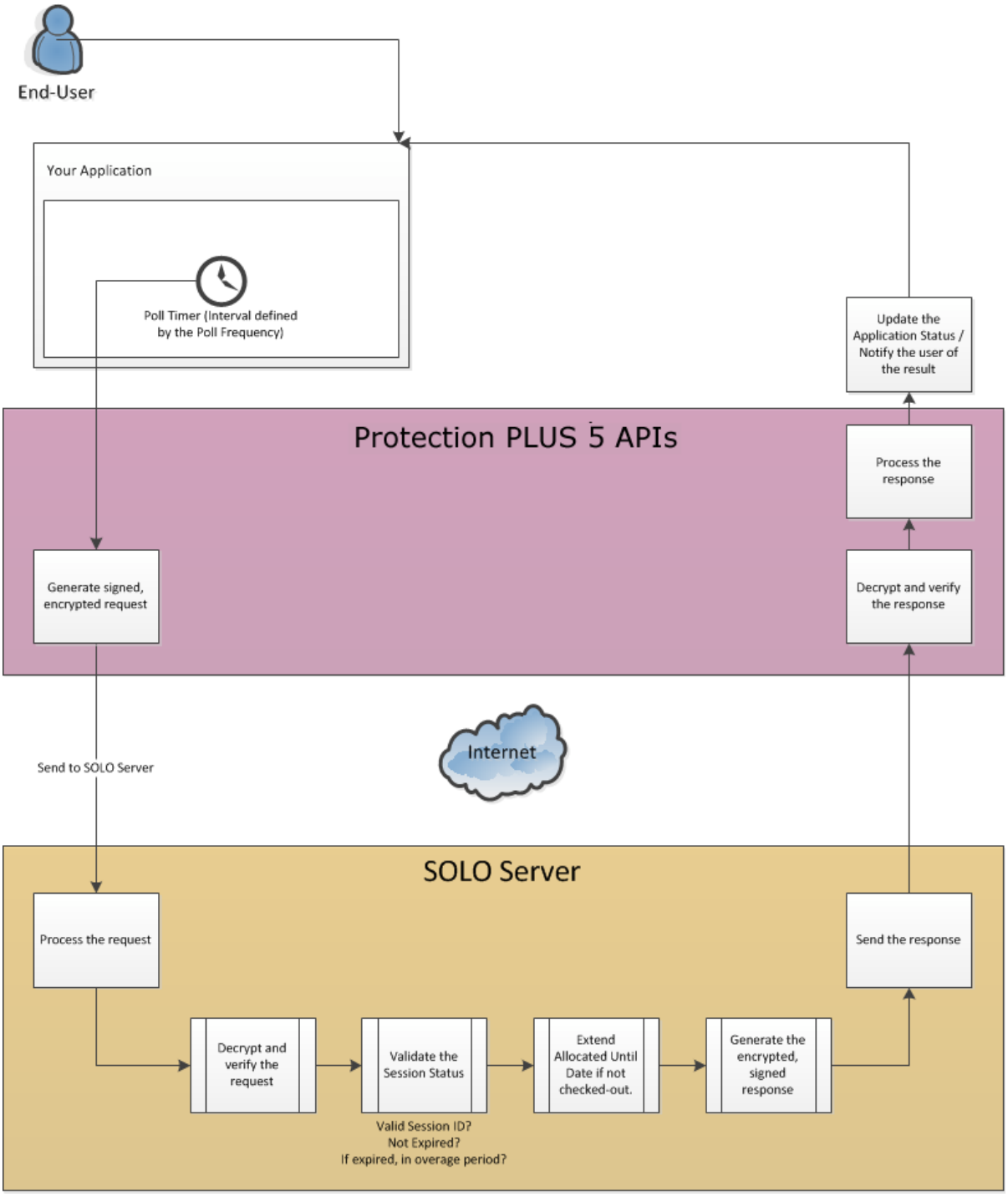
Open

This opens/begins a new network session, resulting in a new, unique Session ID being issued if a seat is still available. When a new network session is opened, it is only valid until its next poll is required.


End-User
Your Application
License ID:
123456789
Password:
****
Open Session
Update the Application Status / Notify the user of the result
Protection PLUS 5 APIs
Process the response
Generate signed, encrypted request
Decrypt and verify the response
Send to SOLO Server
Internet
SOLO Server
Process the request
Send the response
Decrypt and verify the request
Validate the License Status
Make sure enough seats are available
Generate the encrypted, signed response
Valid License ID & Password?
Status is OK?
Not Expired?

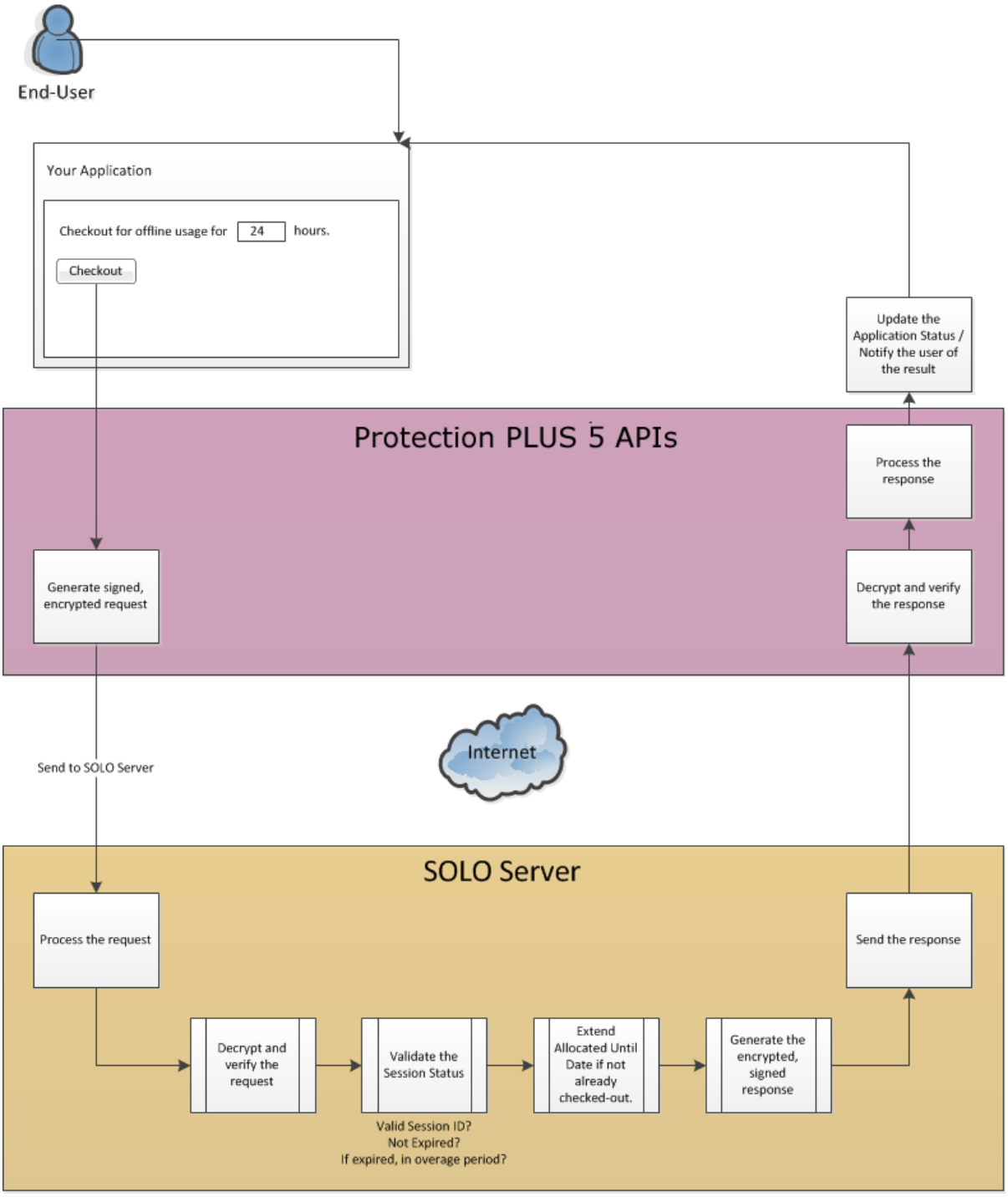
Poll

A poll is where the software checks the status of the current session with SOLO Server. This action prompts SOLO Server to extend the Allocated Until Date until the next poll is required.


End-User
Your Application
Poll Timer (Interval defined by the Poll Frequency)
Update the Application Status / Notify the user of the result
Protection PLUS 5 APIs
Process the response
Generate signed, encrypted request
Decrypt and verify the response
Send to SOLO Server
Internet
SOLO Server
Process the request
Send the response
Decrypt and verify the request
Validate the Session Status
Extend Allocated Until Date if not checked-out.
Generate the encrypted, signed response
Valid Session ID?
Not Expired?
If expired, in overage period?

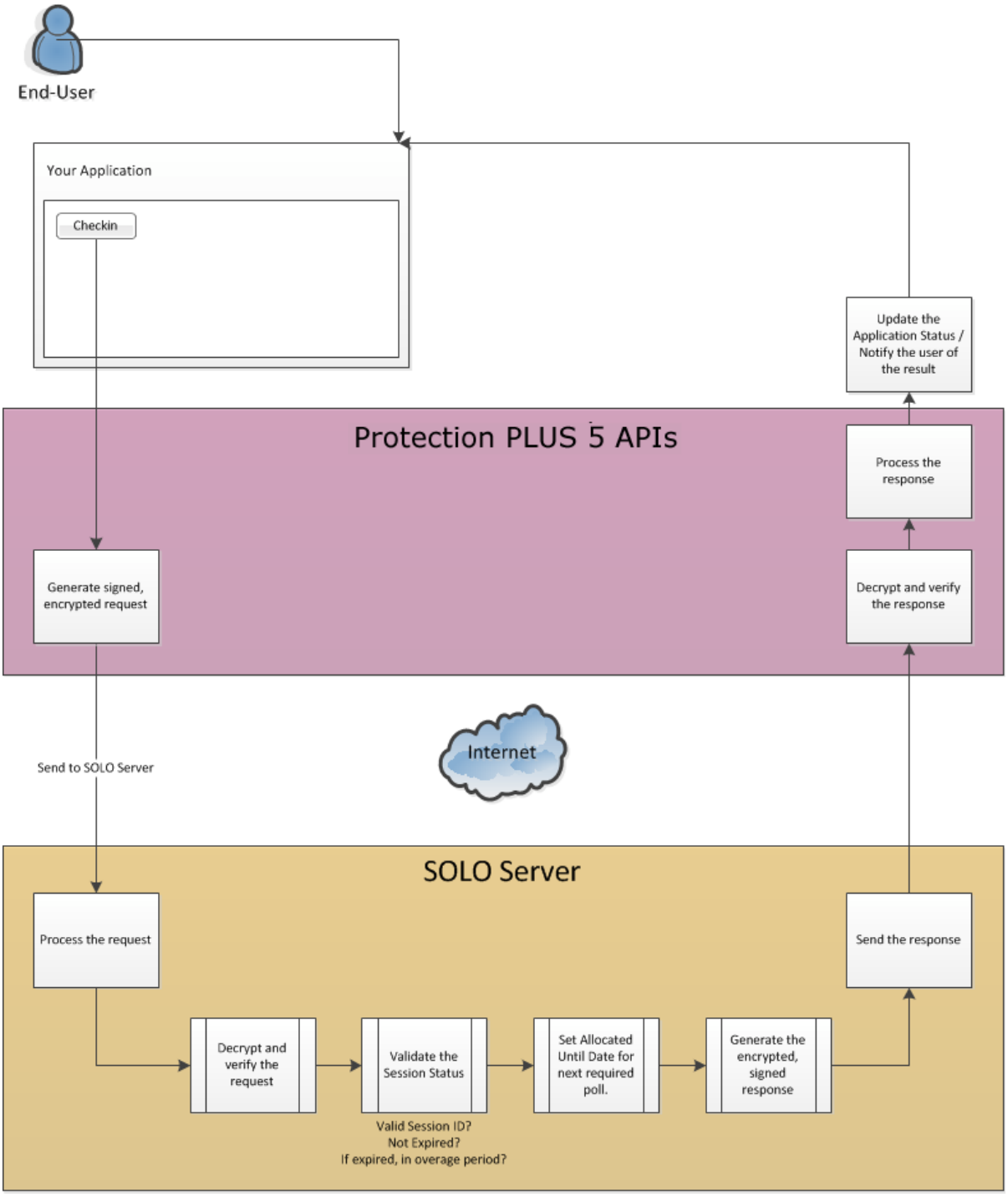
Check-out

If a user needs to use the application in a disconnected state for some time, the user may request a checkout for that duration of time. During the time in which a session has been checked-out, it will consume a seat for the entire checkout duration or until it has been checked-in. It is possible to poll against a checked-out session, but in this scenario, a poll will not extend the session's Allocated Until Date. The availability of this feature will be at the software author's discretion, as would the minimum and maximum checkout durations allowed. Checking-out a session results in a certificate file being created, which allows you to resume a session after exiting and running the application again later. When resuming, the application must load the certificate file from the same path in which it was created. The protected application can also lock the session file when running to prevent other instances of the application from also using it on the same computer.


End-User
Your Application
Checkout for offline usage for 24 hours.
Checkout
Update the Application Status / Notify the user of the result
Protection PLUS 5 APIs
Process the response
Generate signed, encrypted request
Decrypt and verify the response
Send to SOLO Server
Internet
SOLO Server
Process the request
Send the response
Decrypt and verify the request
Validate the Session Status
Extend Allocated Until Date if not already checked-out.
Generate the encrypted, signed response
Valid Session ID?
Not Expired?
If expired, in overage period?

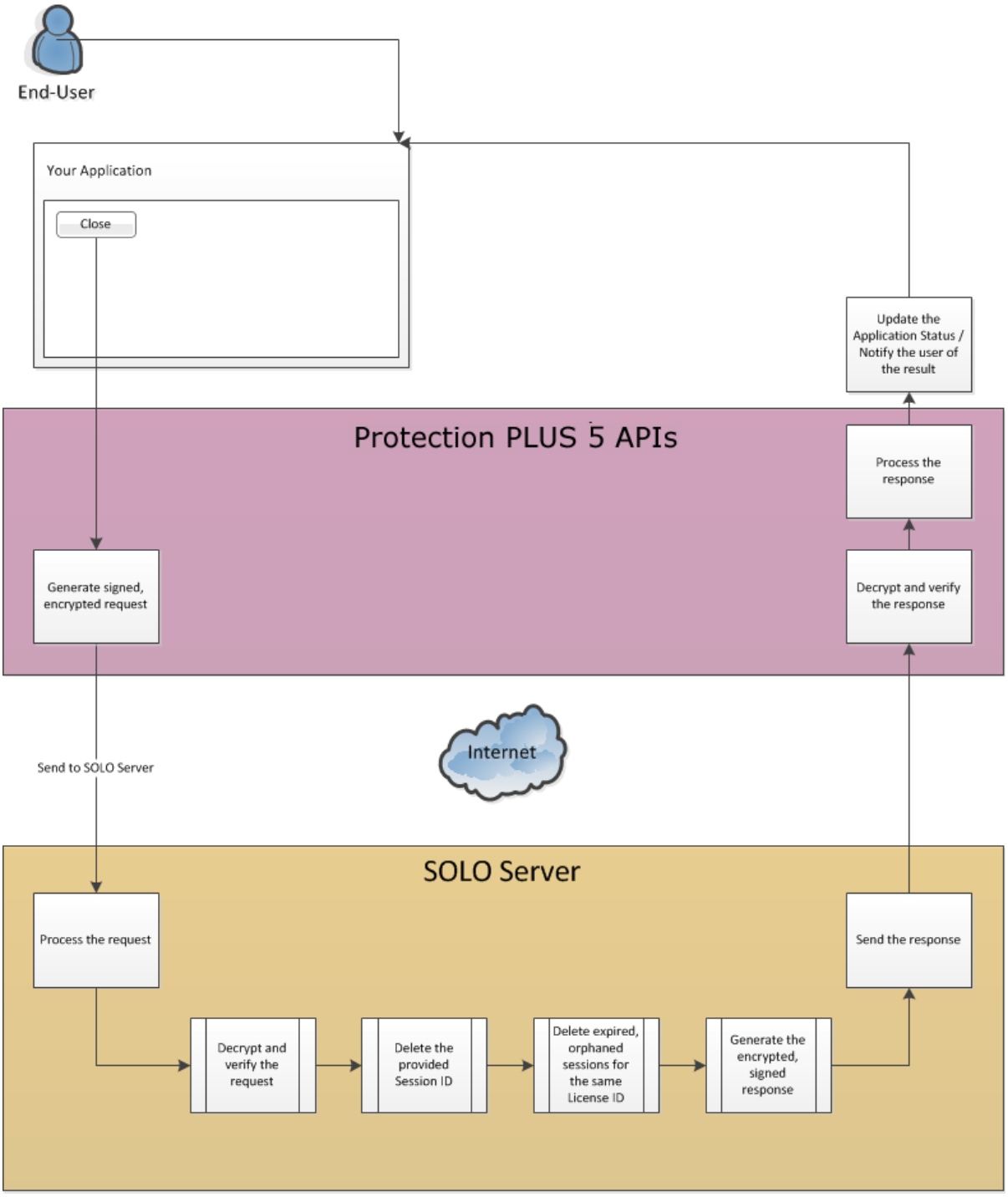
Check-in

When a network session is checked out, its seat is consumed for the full duration of the checkout period unless it is checked-in. After being checked-in, a session may resume its normal polling to remain active. The availability is at the software author's discretion, and it is important to note that **this feature should be used with caution**. Caution is necessary because it is possible for a user to make a backup of the session certificate file, perform the check-in, disconnect, and then restore the session certificate file to resume offline use of the application without consuming a seat as it should. The only means of preventing the user from doing this is to either require the application to poll at least once when restoring a certificate file (usually during application start-up) to verify that the checked-out session is still valid, or to simply not make the check-in feature available to your users. If you decide this feature is necessary, you should take additional measures in your application to prevent users from restoring session certificate files recently checked-in sessions (i.e. you could store these recently checked-in Session IDs in a hidden, encrypted file or registry key).


End-User
Your Application
Checkin
Update the Application Status / Notify the user of the result
Protection PLUS 5 APIs
Process the response
Generate signed, encrypted request
Decrypt and verify the response
Send to SOLO Server
Internet
SOLO Server
Process the request
Send the response
Decrypt and verify the request
Validate the Session Status
Set Allocated Until Date for next required poll.
Generate the encrypted, signed response
Valid Session ID?
Not Expired?
If expired, in overage period?

Close

When a user is done working with a given network session, and the application is being closed, the network session should be closed. Closing a network session makes the current session invalid, and frees the seat the active session consumed so a new session may be opened later.


End-User
Your Application
Close
Update the Application Status / Notify the user of the result
Protection PLUS 5 APIs
Process the response
Generate signed, encrypted request
Decrypt and verify the response
Send to SOLO Server
Internet
SOLO Server
Process the request
Send the response
Decrypt and verify the request
Delete the provided Session ID
Delete expired, orphaned sessions for the same License ID
Generate the encrypted, signed response

Working with Cloud-Controlled Network Floating Licensing using SOLO Server

After **adding a license in SOLO Server**, you can edit the license details to customize additional settings. When editing these details, you can change the number of network seats a given license may allocate by modifying the *License Counter* value.

For more information on implementing Cloud-Controlled Network Floating Licensing with either PLUSManaged or PLUSNative, view one of these manual topics:

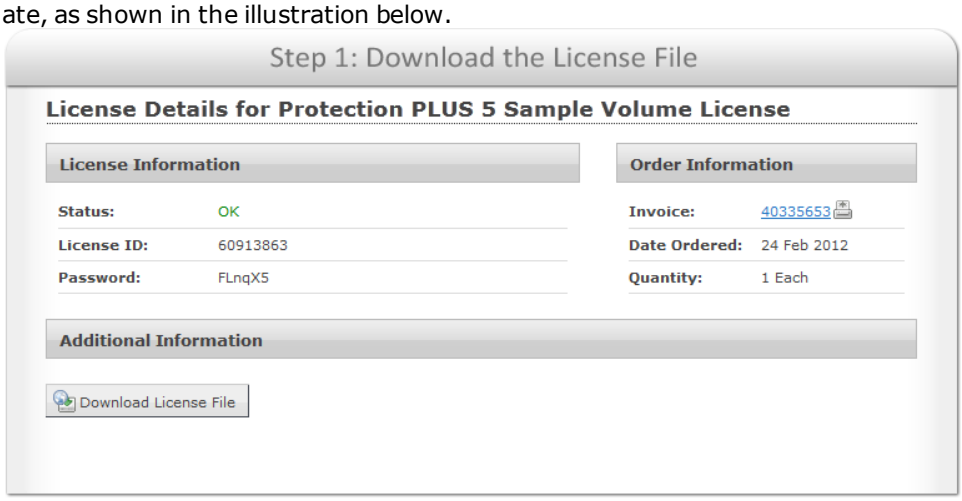
- **PLUSManaged: Cloud-Controlled Floating Network Licensing**
- **PLUSNative: Cloud-Controlled Floating Network Licensing**

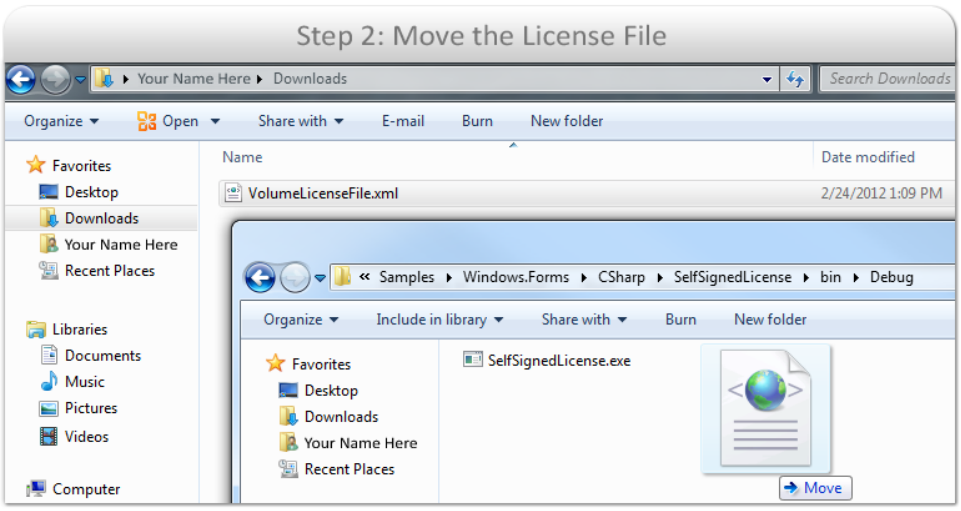
Volume and Downloadable Licenses

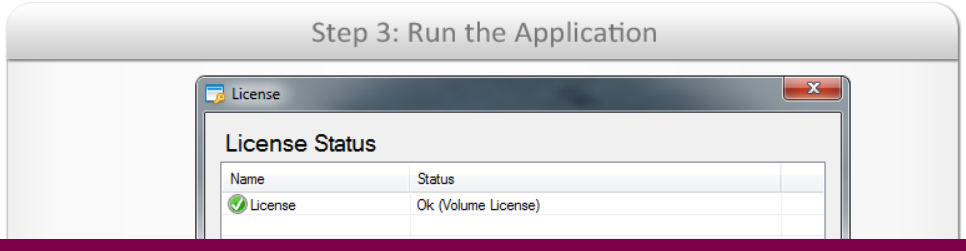
When setting-up your **SOLO Server product configuration**, you can specify the product option type as a Volume License or a Downloadable License with Trigger Code Validation. Your application can distinguish these types of licenses from others by examining the product option type stored **in the license file**.

Volume Licenses

Volume licenses are comprised of a read-only **license file**, which contains data necessary to uniquely identify a license. However, these licenses do not contain any data that uniquely identifies a licensed system. The benefit is that your customers can freely copy and use the volume license file to use your application without the need to activate, as shown in the illustration below.







Since the volume license is completely trusted by your application, it carries the risk that the volume license file could fall victim to unauthorized file sharing over the Internet. If that happens, you would have enough information to identify which license was shared, and you may address this concern through periodic background checking; however, background checking may be impractical to achieve in environments where Internet connectivity is very limited or completely unavailable.

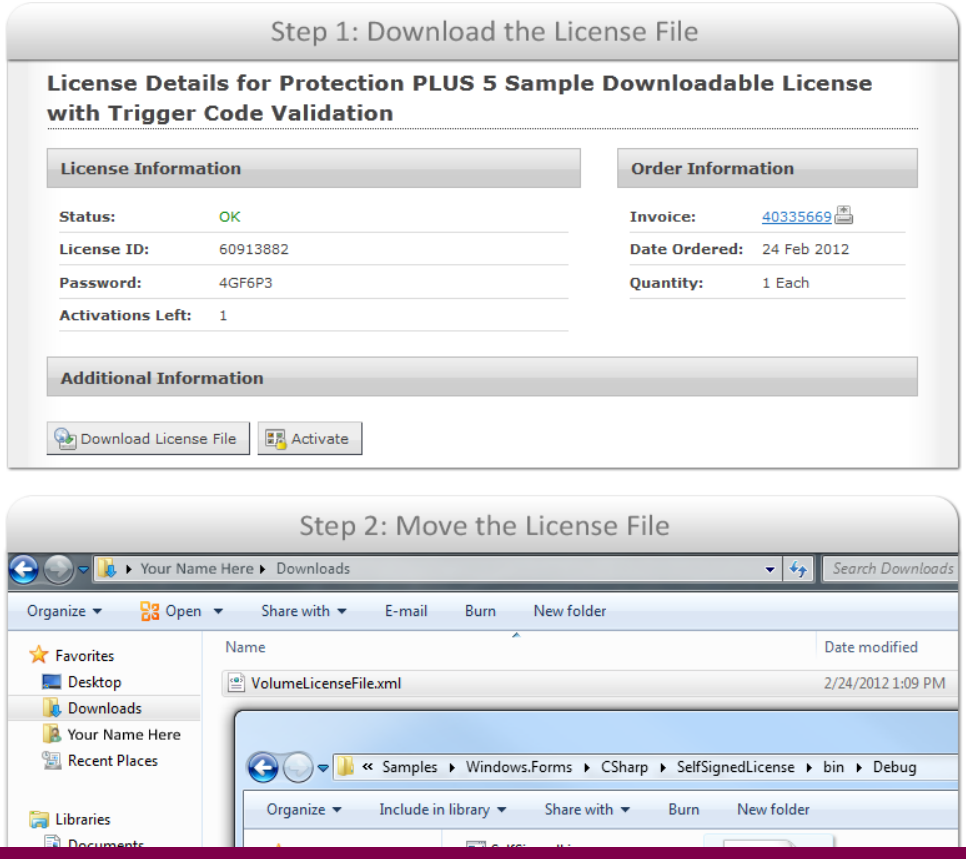
When are Volume Licenses appropriate?

Sometimes, your customers might request a site-wide or enterprise-wide license to install and use your application freely for hundreds or possibly thousands of users, and you and/or your customer have concluded that typical software activation is not an option. This conclusion may be reached when many or most of these users lack a reliable Internet connection, when the corporate IT policy restricts Internet access for many or most users, or any other scenario where requiring activation can pose a serious challenge to deploying and using your application.

In many cases, **network floating licensing** is best suited for licensing a site or enterprise, as it only verifies the application is running on the appropriate network, and only allows a limited number of users to run the application concurrently. However, network floating licensing requires network connectivity, which may also pose deployment challenges when licensing disconnected systems, and/or systems that are spread across many different physical locations. If you have or expect customers which need to be able to easily access your application without activation, then volume licensing may be the solution.

Downloadable Licenses with Trigger Code Validation

Downloadable licenses are very similar to volume licenses, in that they are read-only license files that only contain data capable of uniquely identifying the license. The difference, however, is that these licenses require trigger code validation to activate a separate, writable **license file** on each system on which the application is run, as shown in the illustration below.



What is Trigger Code Validation?

Trigger code validation is a means of achieving software activation through the manual exchange of numeric values (in other words, it is a challenge-response mechanism). Allowing activation through manually exchanged numeric codes is convenient when you need to be able to activate remote systems which lack Internet connectivity, as it is easy to exchange these codes via a telephone call, text messages, etc... Trigger Codes are only capable of carrying a very limited, numeric payload, but this limitation is easily overcome by coupling trigger code validation with the downloadable license file (which can contain any and all license data). Keep in mind that any challenge-response mechanisms such as trigger code validation can be reverse engineered (similar to how "key generators" are available on the Internet for many popular software applications).

When are Downloadable Licenses appropriate?

Downloadable licenses are designed to give you all of the benefits of the rich, extensible Protection PLUS 5 SDK **license file** format, while giving you the ability to easily activate customers who need to use your application in locations that completely lack Internet connectivity. This gives your customers the flexibility of running your applications anywhere, while giving you the peace of mind that unauthorized copies cannot simply be shared through unauthorized file sharing.

Distribution

Volume and downloadable licenses are typically distributed by license files downloaded through SOLO Server's customer license portal. Using your SOLO Server account, you can download volume and downloadable license files through this interface, which can be convenient when shipping the licensed software with physical media and/or related devices or appliances. Alternatively, you may instruct your customers to sign-in to SOLO Server's customer license portal to download the volume license file. Another option is to allow your customers to receive a volume license file by activating over the Internet, though this option may not be feasible in environments with limited or no Internet connectivity.

Defining Licensing Requirements

Before beginning source code integration, it is important to define the application's licensing requirements. Several questions are listed below, which are designed to help you establish a list of requirements for each of your applications.

In order to best identify your licensing requirements as outlined below, it is often best to start with a complete list of use cases. In other words, list out all the different ways you wish to allow someone to purchase, use, license, activate, manage the license, etc... If you wish to take a more illustrative approach, you can also story-board the process using rough wire-frame representations of what you would expect users to see in your application.

Skipping the use case brainstorming scenario is likely to bring you the burden of more work in the future. The SoftwareKey Professional Service Team is happy to work with you through a discovery process to make sure all use cases have been listed and thought through. We can help you by asking questions and discussing advantages and disadvantages of each licensing approach. Please **contact us** for assistance.

Basic Requirements

- What are all of the possible ways customers will purchase licenses to my software?
  - An online store with a credit card and automated license delivery
  - An offline purchase by another method and manual license delivery
- What **types of licenses and restrictions** must be applied for your application? Some examples include:
  - Evaluation (also known as trial, demo, or shareware)
  - Periodic (also known as lease or subscription)
  - Non-expiring
- For each type of license which may be issued, should the license expire, and if so, why and when should this occur? Here are some arbitrary examples:
  - Evaluation licenses should expire after 30 days.
  - Evaluation licenses should expire after 10 uses, or 7 days, whichever occurs first.
  - Subscriptions may be purchased, which last 90 days. After 90 days, a subscription renewal may be purchased to extend the license subscription an additional 90 days.
- What are all of the possible ways customers should be able to activate / deactivate / transfer licenses?
  - Will Internet connectivity be required on the system being activated.
  - If Internet connectivity is not available on the system being activated, is it practical to use another system which has Internet access to activate the target system? (This is possible to do by transferring files via removable media, like a USB drive.)
  - If Internet connectivity is not available, how else should users be able to activate? (For example, telephone, fax, etc...)
- How reliable is Internet connectivity for your users? Contributing factors which can impact this includes (but are not limited to):
  - Internet connection speed and quality. Mobile, dial-up, and satellite connections may be slower or less reliable than other types of connections.
  - Personal firewall software, which may block individual applications, and require users to occasionally configure an exception.
  - Network firewall configurations may be configured with very restrictive policy.
  - Proxy servers sometimes require additional configuration by the end-user, and may also be configured with very restrictive policy.

Advanced Requirements

- Will your application allow users to **electronically manage their licenses**? This can include actions such as deactivating or transferring activated licenses, which is only supported when activation through SOLO Server is used, and therefore requires Internet connectivity either directly or through a nearby machine.
- Will status checking, or phoning-home, be supported or required? (This should be required if users are allowed to deactivate or transfer activated licenses.)

    - How often will your application try to perform status checking?
    - How often will your application require status checking?
- Does your application have modules and/or features that need to be licensed as well? If so...
    - Does each share the application's license? If so, do they only need to be toggled (enabled/disabled) conditionally?
    - Does each instead have its own license (meaning these questions should be asked separately for each applicable module or feature)?
- Should customers be able to purchase software maintenance renewals online?
    - What are the pricing rules when their maintenance is current versus expired?
    - How long after expiration should they be able to renew before paying a reinstatement fee or being forced to purchase a new license instead.
- Does your application consist of a suite of applications? (This can be similar to the modules and/or features described above.)
- Does your application run in a networked environment, where a group or groups of users share a single license?
- Does your application need to support pay-per-use restrictions, and require reactivation after each use or several uses?
- Should customers with expired software maintenance be able to download minor version updates? Major version updates? For how long after the maintenance expires?

Selecting a Solution

Application Characteristics

The SoftwareKey System™ is a large, flexible system, which makes a great deal of options available in order to provide the best solutions possible for simple licensing needs, complex licensing needs, and everything in between. If you find that selecting the correct solution is confusing or overwhelming, **contact us** for guidance. Otherwise, the first step is to identify some key characteristics of your application, which you can accomplish by answering some or all of the questions below. If you are not a developer, then this is something you can send to your developer(s) to get answered.

- Is the application a .NET/managed application (written with C# or VB.NET), or an application written in native code (such as C/C++).
- What is the nature of the application which needs to be licensed? For example, is it a standard Windows GUI application, component, service, web application, etc...?
- How critical is security and integrity of the license?
- How critical is the availability of your application to its users? This can impact decisions on how or when to prompt and/or lock-out users when license validation fails.

In addition to the questions above, it is also helpful (and often necessary) to know your application's **licensing requirements** to select the most appropriate solution.

Instant Protection PLUS 3

Protection PLUS 5 SDK offers robust application programming interfaces (APIs), which are typically simple to use. However, Instant Protection PLUS 3 is an alternative which provides a wizard-based interface, and requires little to no source code changes. You should consider **Instant Protection PLUS 3** under the following conditions:

- The application is a standard Windows GUI application (or is not a service or web application, and is not a component that should not show Windows GUI dialogs).
- The application does not have many or any requirements which would require a customized implementation.
- The application does not need to support volume licensing.
- The application does not need access to highly customized license data (such as custom string or XML formatted data).
- The application is a native application (written in C/C++), a .NET/managed application, or consists of a mix of both native and managed code.
- The application licensing dialogs do not need to be customized, and do not need to support UNICODE characters.

Protection PLUS 5 SDK

If Instant Protection PLUS 3 does not suit your needs as described above, or if you simply prefer more control, one of the Protection PLUS 5 SDK application programming interfaces (APIs) is the best choice.

Selecting the Correct Edition(s)

Protection PLUS 5 SDK offers many editions to choose from, and the edition(s) you need will vary based on what language (or combination of languages) you use to write your application(s), and what platforms you need to support.

Edition	Designed for...
.NET Edition	.NET Framework (2.0 or later) or Mono (2.8 or later) applications (written in C#, Visual Basic .NET, etc...) that run in Windows, macOS, Linux. Note that this does not include mobile or embedded device support.

Native Edition for Windows	Native applications (writen in C/C++, Delphi, etc...), or applications that can call native C APIs,/Dynamic Link Libraries (*.dll files), that run in 32 bit (x86) and 64 bit (x64) versions of Windows.
Native Edition for macOS	Native applications (written in C/C++, Delphi, Objective C, Swift, etc...), or applications that can call native C APIs/Dynamic Libraries (*.dylib files), that run in macOS.
Native Edition for Linux	Native applications (written in C/C++, Delphi, etc...), or applications that can call native C APIs/Shared Object libraries (*.so files), that run in 32 bit (x86) and 64 bit (x64) versions of Linux.
LabVIEW Edition	LabVIEW 2014 or later applications that run in 32 bit (x86) or 64 bit (x64) versions of Windows, macOS, or Linux.
Java Edition	Java 7 or later applications that run in 32 bit (x86) or 64 bit (x64) versions of Windows, macOS, or Linux.
Android Edition	Android applications that run in Android 4.4 (KitKat) or later that run on 32 bit ARMv7 or x86 devices.

If you still are not sure what edition(s) may be right for you, **contact us**. Likewise, **contact us** if you do not see an item listed above that fits your requirements (e.g. you use another language, platform, architecture), as we may still be able to fulfill your licensing needs.

Understanding the APIs

Once you have acquired the correct edition(s), you can then begin **reviewing the samples** for the APIs (described below) that apply to you.

PLUSManaged

If your application is written using the .NET Framework (C# or Visual Basic .NET), then **PLUSManaged** is the rich, object-oriented, fully managed API designed for you. This applies to Protection PLUS 5 SDK .NET Edition.

PLUSManagedGui

Creating user interfaces to allow users to manage your application licenses involves design, development, and test-ing. PLUSManagedGui is a fully managed, .NET component that provides you with an easy-to-use, pre-built GUI that you can simply plug right into your applications. PLUSManagedGui is designed for use with applications that use PLUSManaged and are built on or able to use `System.Windows.Forms` GUIs (PLUSManagedGui is not designed to be integrated into services or web applications directly). This applies to Protection PLUS 5 SDK .NET Edition.

PLUSNative

If your application is anything other than a .NET application, or is written in a language which can make calls to shared/dynamic-link libraries, then **PLUSNative** is the API designed for your needs. This applies to Protection PLUS 5 SDK Native, LabVIEW, Java, and Android Editions.

Protection PLUS 5 SDK Sample Applications

The Protection PLUS 5 SDK APIs contain a wide variety of licensing features. There are an assortment of samples available to help demonstrate how licensed applications typically work, how the different licensing features work, and how the APIs may be integrated into your application.

These sample applications are installed with Protection PLUS 5 SDK in archive file format so they may be re-extracted if necessary. On Windows, they can be found by running Protection PLUS 5 SDK and clicking on the *Samples* link, which will open Windows Explorer to the *%PUBLIC%\Documents\SoftwareKey\Protection PLUS 5\Samples* directory. On other operating systems, you will choose where to extract the development files during installation, which will include the samples.

Samples for .NET Applications

Important

If you copy and paste any sample code into your own application, make sure your application's copy of the code is **configured for your SOLO Server account**.

In the *%PUBLIC%\Documents\SoftwareKey\Protection PLUS 5\Samples* directory, you will find solution files for each supported version of Visual Studio, which reference all of the .NET sample projects.

When running a sample project, you may notice Visual Studio starts ASP.NET Developer Server, even when you are not explicitly running one of the ASP.NET samples. To prevent this, you can select the ASP.NET sample projects in Solution Explorer, open the properties pane (or press F4), and change the "Always Start When Debugging" setting to False.

PLUSManaged Samples

The **PLUSManaged samples** show you how you can integrate the PLUSManaged API with any of your .NET applications. Note that these samples lack integration with PLUSManagedGui that is included with the newer samples outlined below. Even though most of these samples do use Windows Forms, it is simple to re-use most of this sample code with nearly any type of interface (as shown with the ASP.NET samples that are included). These samples are especially useful when your application (such as web applications, services, etc...) cannot use a traditional dialog/form based interface.

PLUSManagedGui Samples

The **PLUSManagedGui samples** are similar to the PLUSManaged samples, but they also integrate with PLUSManagedGui which is a component that includes a graphical user interface (GUI) for licensing. This means that these samples (and applications based off them) need less code for typical licensing GUI functionality. Consequently, these samples are also designed to better separate the licensing implementation from the application's user interface. The end result is sample code that is easier to follow and easier for you to re-use in your own applications.

Samples for Native Applications

The **PLUSNative samples** show you how you can integrate the PLUSNative API with your applications. These samples include support for a variety of popular languages and GUI frameworks, including (but not limited to) C++ with MFC, C++ with wxWidgets, Delphi, Objective C, and Visual Basic 6.

Running the Sample Applications

The sample applications are only shipped as source code, so each sample must be compiled before it may be launched. Once compiled, the output executable/binary (or .EXE file) will be located in the project directory. For

.NET applications, this is under the *bin\Debug* or *bin\Release* sub-directory (depending on which build configuration you selected before compiling). For native applications, this varies upon the target architecture (such as *Win32* or *x64*) and the selected build configuration (*Debug* or *Release*). You may either run the samples from by double-clicking the executable/binary file in the output directory, or by clicking a "Run" or "Start Debugging" menu option or toolbar button in your integrated development environment (IDE - e.g. Eclipse, Visual Studio, etc...).

Testing Activation and ELM

For more information about testing activation and **Electronic License Management**, read more about **using SOLO Server** and **creating licenses**.

Important

The sample applications are designed to require activation, just as your protected application would after integration is complete. Before evaluating and running the sample applications, you should become familiar with **how to add test licenses in SOLO Server**.

Clearing License Files and Aliases

It is often necessary to run or debug the application in a scenario where it would behave like it is running on a computer for the very first time. By deleting all of the license files and aliases, you can quickly return the application to a state expected when it is run on a computer for the very first time. By default, the sample applications are configured to store the files in the same directory as the executable (or .EXE) file, while using file names such as *LicenseFile.xml*, *VolumeLicenseFile.xml* (for downloadable and volume licenses), *LicenseAlias1.xml*, and *LicenseAlias2.xml*. These locations are defined in the sample application source code. For example, with PLUSManagedGui samples, you should refer to the `LicenseConfiguration` class, which contains properties (`Aliases`, `LicenseFilePath`, and `VolumeLicenseFilePath`) that define where these files are saved. With PLUSNative samples, you should find and evaluate source code which calls functions like `SK_PLUS_LicenseFileLoad` and `SK_PLUS_LicenseAliasAdd` to determine where these files are stored.

PLUSManaged Sample .NET Applications

The PLUSManaged samples, which lack integration with **PLUSManagedGui** (which you should consider using if you are protecting a traditional form/dialog based application), show you how you can **integrate the PLUSManaged API** with any of your .NET applications. Even though most of these samples use Windows Forms, it is simple to re-use most of this sample code with nearly any type of interface (as shown with the ASP.NET samples that are included). These samples are especially useful when your application (such as web applications, services, etc...) cannot use a traditional dialog/form based interface.

Overview

Here are some of the features and highlights to note about the PLUSManaged sample applications:

- The collection of basic samples included are each named after the type of license(s) supported. These are described in the Sample Projects section below.
- For the samples that use **writable licenses** (the "SelfSignedLicense" and SelfSignedTrial" samples), a 30-day evaluation is automatically started the first time the application runs on a computer.
- Activated licenses do not expire.
- These samples allow users to **activate** using a direct Internet connection, or manually using another device's Internet connection.
- Using a direct Internet connection, users can **deactivate** a previously activated license, which allows your application's users to essentially transfer the license to another computer. This is a convenient way to allow your customers to migrate to a new computer, without the need to contact you for support or additional activations. SOLO Server can limit the number of times a license is deactivated, and retains the history so you can make informed decisions when providing support.
- The application can **validate and refresh** its license with SOLO Server using a direct Internet connection.
  - This can be done on demand, which is especially useful if you expect users may need to have a way to update their license properties quickly. (E.g. adding extra network seats, extending/renewing a time-limited license, etc...)
  - This can be done automatically, which is important so that your application can detect when it has been deactivated previously or remotely.
  - You can also control how frequently your application tries to validate in this manner, and how frequently it is required to validate in this manner. This prevents network congestion, and helps avoid the perception that your application is slow or unresponsive should an unreliable network connection cause such delays. Further, when offering time-limited licenses that can be renewed/extended, this is a convenient way to automatically retrieve the new expiration date after the renewal has been processed.
- These samples include the source code for all forms/dialogs used, which makes it easy for you to customize in any way needed.
- These samples also include basic support for proxy servers and proxy authentication.
- The "SimpleTextEditor" samples are designed to show what a licensed application would look like. It is intentionally similar to what you would expect from a basic text editor that ships with your operating system.
  - These samples also show you how you can enable and disable individual features for licenses. In this case, these samples conditionally enable/disable menu and toolbar buttons based on what bits are enabled on the TriggerCodeFixedValue property (set in the Product Option in SOLO Server) or the UserDefinedNumber1 property (set in the user defined fields for the License ID in SOLO Server).
- The NetworkFloatingSemaphore samples show how to limit the number of concurrent users (or instances of your application) that can use your protected application using exclusively locked semaphore files on a Windows (SMB) share.

To see additional licensing features in action, check out the **PLUSManagedGui samples**!

Sample Source Files

The *%PUBLIC%\Documents\SoftwareKey\Protection PLUS 5\Samples* directory can be opened by running Protection PLUS 5 SDK from your start menu/screen, and clicking on the *Protection PLUS 5 SDK Samples* link. This

directory contains all of the files and folders listed below, as well as several solution files (one for each supported version of Visual Studio). Each solution is configured with the sample projects that are designed for the corresponding version of Visual Studio, and the projects are organized by the underlying framework and language used to develop each sample application. For example, any C# applications that use `System.Windows.Forms` dialogs are located in the *CSharp\Windows.Forms* directory (in the Solution Explorer pane/tab).

Important

Many of the source code comments begin with `TODO` and `IMPORTANT`. **It is very important that you review each of these comments in any and all files you copy into your application.** Each of these comments indicates an area where you need to make an informed decision about how the application should behave and react in certain scenarios. Disregarding these comments can lead to undesirable behaviors in your application.

Sample Projects

Each project listed below is contained in its own directory, and each of these directories is prefixed with "PLUSManaged_" in order to distinguish them from the PLUSManagedGuisamples. When you open the sample solutions however, these samples are isolated under a PLUSManaged solution folder (under the solutions folders for the respective language and presentation/GUI framework).

Type	Sample Projects	Description	Supported Platforms/Frameworks
Read-Only License Files	ReadOnlyLicense SimpleTextEditorReadOnly	Shows how you can use a read-only **license file** for the highest level of **security**. These samples do not include evaluation licensing, though this is possible to add.	.NET, ASP.NET
Mixed Read-Only and Self-Signed License Files	SelfSignedTrial SimpleTextEditorSelfSignedTrial	Although these samples are a little more complex, they provide a great middle-ground for **security** and flexibility by automatically creating their own, self-issued evaluation licenses (using a writable license file), while using the more secure read-only license files for activated copies.	.NET

Self-Signed (Writable) License Files	SelfSignedLicense SimpleTextEditorSelfSigned	These samples always use a self-signed/writable **license file** for all licenses. These types of license files provide the highest level of flexibility, as they allow your application to freely modify the license file (even without Internet connectivity). However, using writable/self-signed licenses means the protected application uses key data fully known to it when encrypting license files, which is less **secure** than read-only licenses (which use key data only partially known to the protected application).	.NET
Network Floating via Semaphores	NetworkFloatingSemaphore	Shows how you can use semaphore files (or files locked to a running instance of your application) on a Windows (SMB) share to **limit the number of users or instances of your application** on a network.	.NET
Cloud-Controlled Floating Network Licensing using SOLO Server	CloudContolledNetworkFloatingSOLOServer (**contact us** for details)	Shows how you can leverage SOLO Server to **limit the number of concurrent users or instances of your application**. This requires Internet connectivity (at	.NET

		least initially), and includes advanced features not easily achieved via semaphore files (such as checking-out a seat for later use when offline/disconnected).

Source Code Files

The PLUSManaged samples use a variety of source code files that are common amongst many or all of the sample applications. The source files are summarized below, and each source file contains commenting which documents the source code in great detail. A summary of the shared source code files is provided below.

Class/Type	File Locations	Description
AboutForm	*AboutForm.cs*, *AboutForm.Designer.cs*, *AboutForm.vb*, *AboutForm.Designer.vb*, *AboutForm.resx*	Used only in the SimpleTextEditor samples, this form displays the status of the application's license, and allows the user to activate, deactivate, and refresh his or her license.
Feature, Features, LicenseFeatures	*Feature.cs*, *Feature.vb*, *Features.cs*, *Features.vb*	Used only in the SimpleTextEditor samples, these include the classes and methods necessary to define and validate individual application features.
FindAndReplaceForm	*FindAndReplaceForm.cs*, *FindAndReplaceForm.Designer.cs*, *FindAndReplaceForm.vb*, *FindAndReplaceForm.Designer.vb*, *FindAndReplaceForm.resx*	Used only in the SimpleTextEditor samples, this form allows the user to search for and optionally replace text that resides in the MainForm.
LicenseConfiguration	*LicenseConfiguration.cs* *LicenseConfiguration.vb*	Contains licensing configuration properties, including settings required for encryption, and settings which define how the application should behave and react in various circumstances. It is **very important** to review all of the TODO and IMPORTANT comments throughout this source file. This file contains code that requires changes before using it with your application.
MainForm	*MainForm.cs*,	Contains the main form or dialog

	MainForm.Designer.cs, *MainForm.vb*, *MainForm.Designer.vb*, *MainForm.resx*	implementation for the sample applications. This dialog is designed to provide a simple way of showing how your applications can interact with the licensing objects.
`ManualActivationForm`	*ManualActivationForm.cs* *ManualActivationForm.vb* (Has other supporting files.)	Contains the form or dialog implementation that allows the protected application to be activated manually. This gives your application's users a means to activate a computer that does not have a direct Internet connection. This does, however, require the user to use another device that does have Internet connectivity. If your application needs to support activation without any Internet connectivity at all, then you should consider using **trigger codes**, which are supported in the **PLUSManagedGui samples**.
`OnlineActivationForm`	*OnlineActivationForm.cs*, *OnlineActivationForm.Designer.cs*, *OnlineActivationForm.vb*, *OnlineActivationForm.Designer.vb*, *OnlineActivationForm.resx*	Contains the form or dialog implementation that allows the protected application to be activated using a direct Internet connection via SOLO Server.
`ProxyCredentialsForm`	*ProxyCredentialsForm.cs*, *ProxyCredentialsForm.Designer.cs*, *ProxyCredentialsForm.vb*, *ProxyCredentialsForm.Designer.vb*, *ProxyCredentialsForm.resx*	Contains the form or dialog implementation that prompts the user to enter his or her proxy server authentication credentials. This prompt is only displayed when proxy server authentication is required, and the protected application is attempting to contact SOLO Server using an Internet connection for the first time. (The credentials are automatically cached by `WebServiceHelper` as long as the protected application is running.)
`SampleLicense`	*SampleLicense.cs* *SampleLicense.vb*	Implements the `License` class or the `WritableLicense` class as appropriate for the given sample (for read-only, or writable **license files**, respectively). In the self-signed trial samples, this is an interface that allows the sample to perform actions on either type of license file.

SampleReadOnlyLicense	*SampleReadOnlyLicense.cs* *SampleReadOnlyLicense.vb*	Implements the License class in PLUSManaged and provides an implementation that only uses read-only **license files**. This implementation is used for activated licenses in the Self-Signed Trial samples.
SampleSelfSignedLicense	*SampleSelfSignedLicense.cs* *SampleSelfSignedLicense.vb*	Implements the WritableLicense class in PLUSManaged and provides an implementation that only uses writable/self-signed **license files**. This implementation is used for evaluation/trial licenses in the Self-Signed Trial samples.
SplashScreenForm	*SplashScreen.cs*, *SplashScreenForm.cs*, *SplashScreenForm.Designer.cs*, *SplashScreen.vb*, *SplashScreenForm.vb*, *SplashScreenFormDesigner.vb*, *SplashScreenForm.resx*	Used only in the SimpleTextEditor samples, this contains the form or dialog implementation that shows a splash screen is loading. This example dialog uses System.Threading.Thread.Sleep to create an artificial delay to simulate what it would look like if your application ran a large amount of initialization logic while displaying the splash screen. It also initializes the features that are enabled and disabled in the SimpleTextEditor sample application.
WebServiceHelper	*WebServiceHelper.cs* *WebServiceHelper.vb*	Used in the sample Windows Forms applications, this class simplifies calling web services. It contains the default SOLO Server URLs used by the protected application, and is also what handles proxy server support (and proxy server authentication) in the sample applications.

Configuring your licensing options

The LicenseConfiguration class contains a variety of settings, many of which are extremely important to change before re-using the sample source code in your applications. As noted earlier, many of the source code comments in this file begin with TODO and IMPORTANT. These comments indicate an area of code or a setting that you need to review and, in many cases, update. These settings are outlined in the table below.

Setting Name(s)	Description
Encryption Settings	

`EncryptionKey`	This contains the encryption key data, which usually comes from **your SOLO Server account**. **It is very important that you update this to use your account's data before distributing your application!** If you are evaluating Protection PLUS 5 SDK, also make sure you update the Encryption Key to a non-expiring key which will no longer have the "_EVALUATION_EXPIRES_2013-10-05_" expiration date at the beginning (the actual date will be different). **If you distribute your application with an evaluation key, it will expire and your customers won't be able to use the software.** If you have purchased a license for Protection PLUS 5 SDK but your Encryption Key still has an expiration date, please **contact us**. Additionally, it is important to use the correct key store in this property. In most cases, your desktop applications should use the user key store, while other types of applications, such as services and web applications/services, should typically use the machine key store.
Application & Product Settings	
Application Directory	Gets the absolute path to the directory in which the application (executable file) is located. This property is present as a convenience, and is used by other properties when storing other files (such as the license file, aliases, etc...) in the same directory as the sample application.
ThisProductID	Gets the application's Product ID. This value is typically issued when **configuring a product in SOLO Server**. However, if you are not using SOLO Server, this should contain a value that is unique to each application you protect.
ThisProductVersion	Gets the application's version number. By default, this uses the assembly version of the protected application. This value must contain 4 parts, no longer than 5-digits each (e.g. N.N.N.N, where each N is at least 0, and no larger than 99999).
License File & Alias Settings	
`PATH_REGISTRY_LOCATION`	This constant specifies the Windows registry key value (stored under HKEY_CURRENT_USER) that will be used to store the license file path specified with the NetworkFloatingSemaphore samples. When implementing network floating, it is very important to change this key location to something that is unique to each application you protect.
`Aliases`	Gets a list of aliases used with any samples that use writable license files. By default, this stores the aliases in the same directory as the application or license file for convenience. (This makes it easier for you to simulate running your application on a computer for the first time since its easier to delete the license file and aliases this way.) However, it is very important that you pick better locations to hide/obscure these

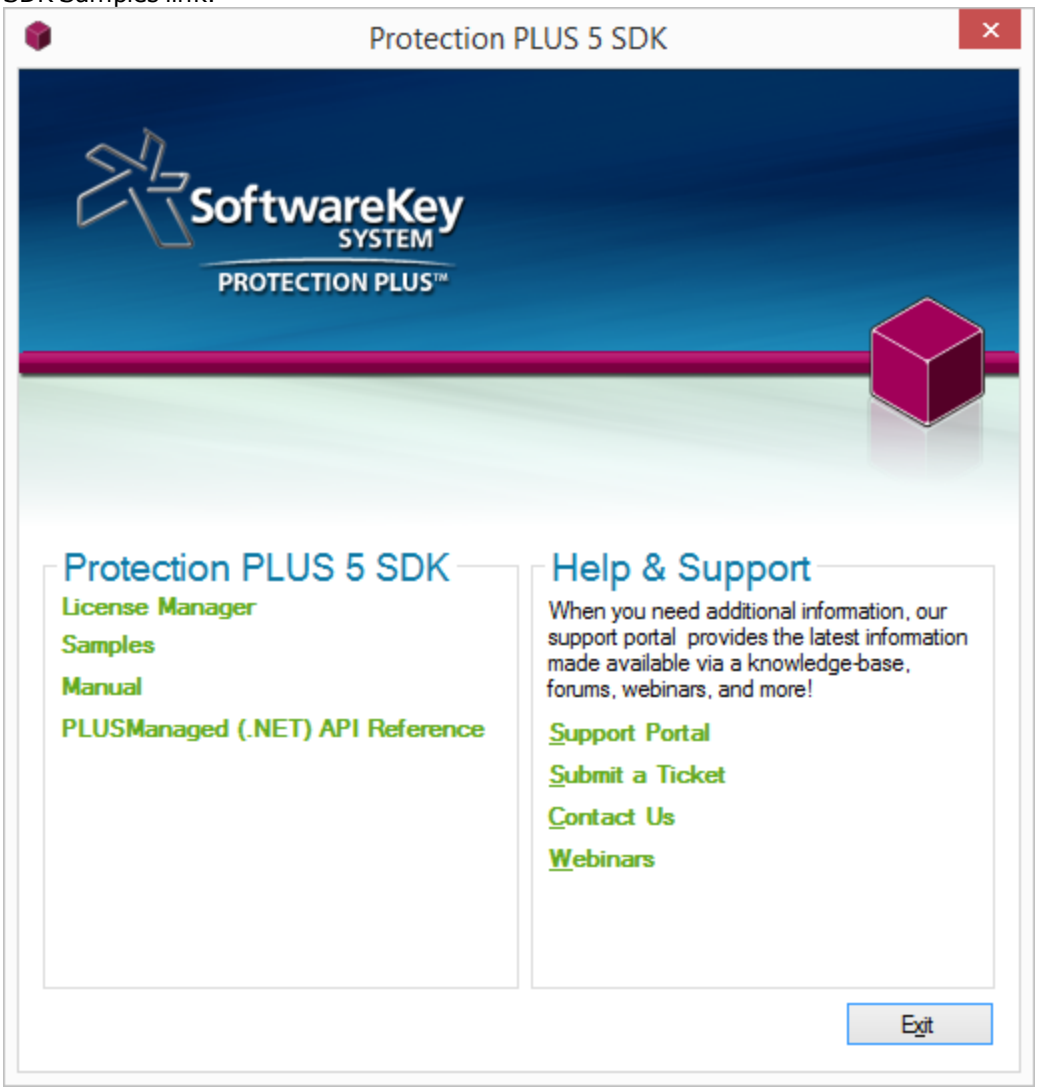
	files for your protected applications. Additionally, it is also very important to make sure the locations selected are unique to each application you protect.
`LicenseFilePath`	Gets the path to the application's license file. By default, this file is stored in the application directory for convenience.
`ManualActionSessionStatePath`	When using PLUSManagedGui, you can configure the component (at design time or run time) to allow one or more actions (such as activation, deactivation, and refresh) to be done manually. This enables your customers to leverage another device's Internet connection to manually activate a system which is not connected to the Internet. This property gets the path to the manual action session state file, which is what stores a manual request so it can be completed at a later time.
`PathRegistryValue`	When using the NetworkFloatingSemaphore samples, this gets or sets the registry value that contains the path to the license file and semaphore files. The registry location is defined in the `PATH_REGISTRY_LOCATION` constant described above.
`NetworkSemaphorePrefix`	When using the NetworkFloatingSemaphore samples, this sets the prefix used for the semaphore file names. So for example, with the default prefix of "sema", the semaphore files created will be named like "sema123.net".
Licensing Restrictions & Settings	
`FreshEvaluationDuration`	When using samples that use a writable license file, this specifies the number of days in which a new evaluation period will last.
`RefreshLicenseAlwaysRequired`	Specifies whether or not the protected application will attempt to validate and refresh its license with SOLO Server every time it starts or validates the license.
`RefreshLicenseAttemptFrequency`	Specifies the number of days to wait before attempting to validate and refresh the license with SOLO Server.
`RefreshLicenseEnabled`	This property is present for convenience, and evaluates RefreshLicenseAlwaysRequired, RefreshLicenseAttemptFrequency, and RefreshLicenseRequireFrequency to determine whether or not license refreshes are enabled at all.
`RefreshLicenseRequireFrequency`	Specifies the number of days to wait before requiring the protected application to validate and refresh its license with SOLO Server.
`SystemIdentifierAlgorithms`	The system identifier algorithms to use to uniquely identify and authorize a system. The identifiers generated by these algorithms are pivotal for **adding copy protection** to your applications.

The `LicenseConfiguration` class will also contain other settings that are not mentioned here because they only apply to the **PLUSManagedGui samples**.

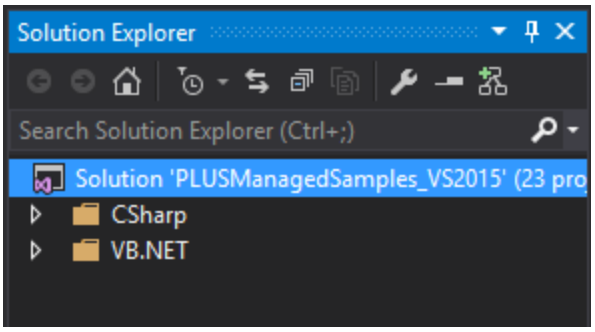
PLUSManaged Licensing Sample

Step 1 – Opening the Licensing Sample Project

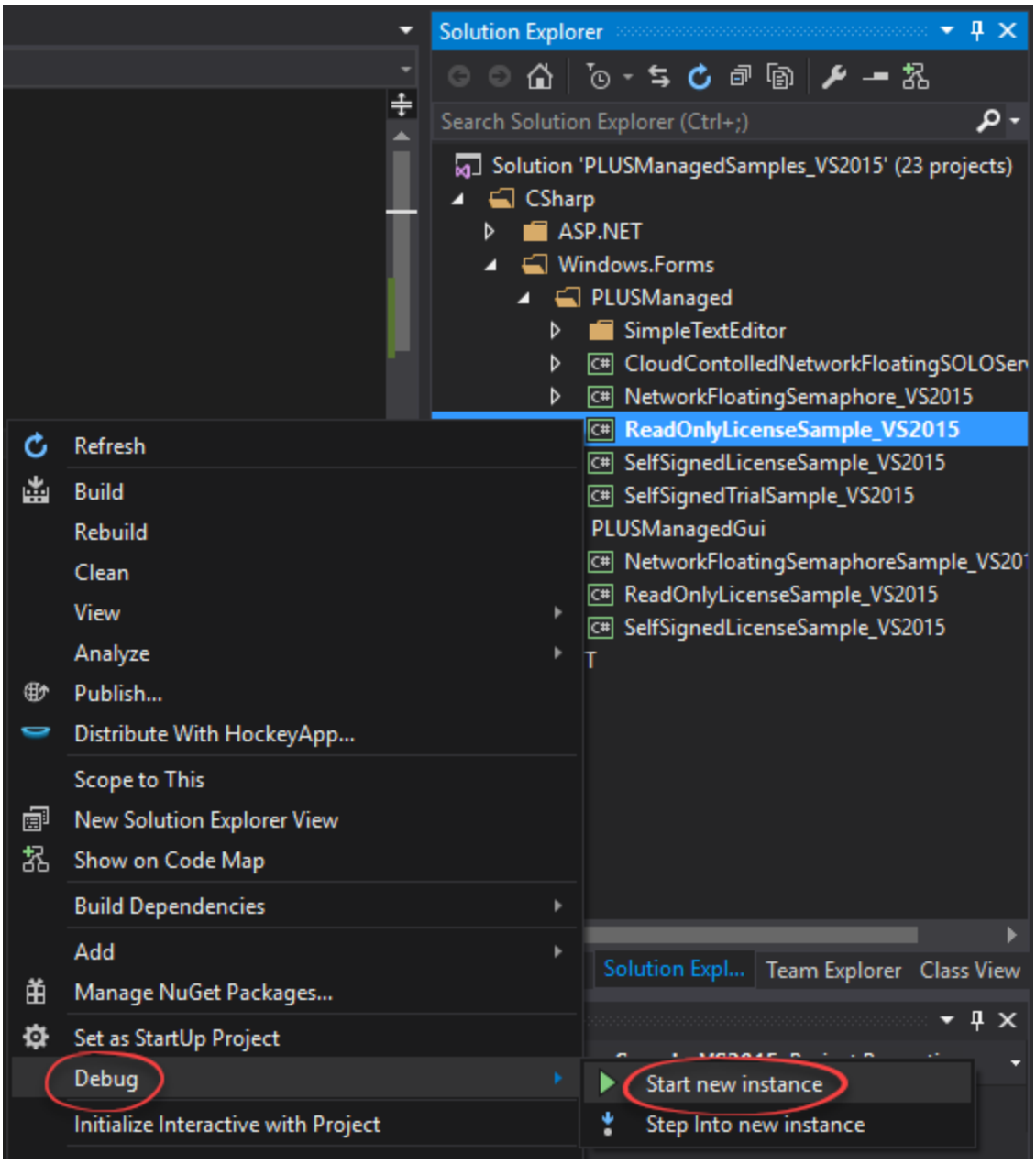
Open the PLUSManagedSamples *.sln file corresponding to your version of Visual Studio. The Samples directory can be opened by running Protection PLUS 5 SDKfrom your start menu/screen, and clicking on the Protection PLUS 5 SDK Samples link.



Once the project has loaded in Visual Studio, expand the programming language you wish to proceed with (C# or VB.NET).

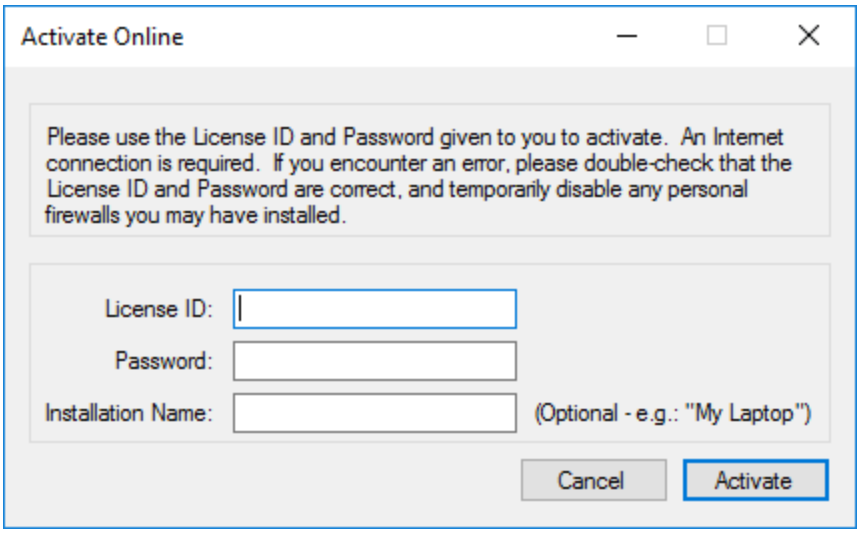


Expand the Windows.Forms folder and PLUSManaged folder. Locate the ReadOnlyLicense Sample. Right click this sample and select Debug > Start New Instance



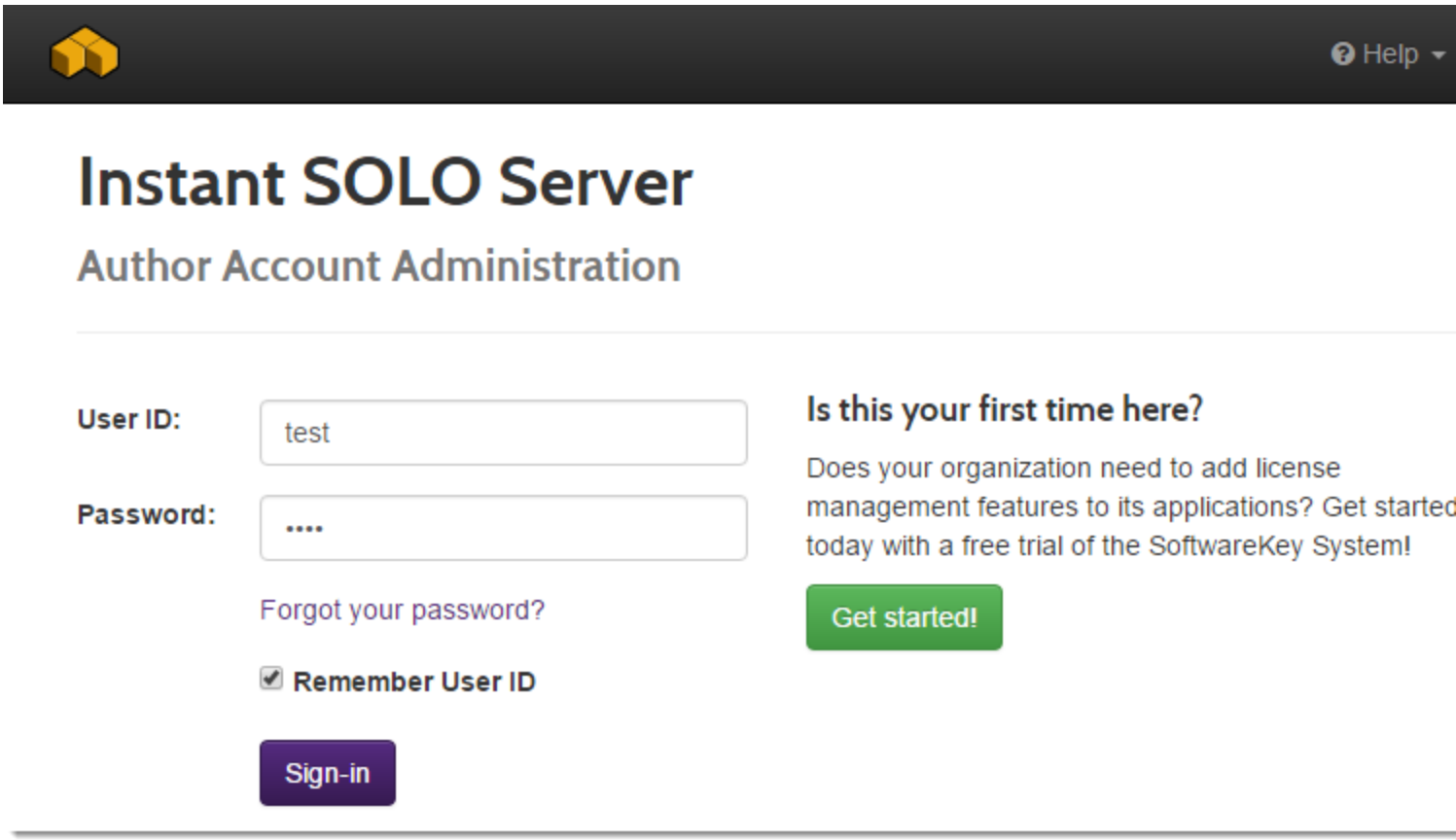
Step 2 – Activating automatically with Instant SOLO Server

To activate the Sample Licensing Application automatically using *Instant SOLO Server*, click on Activate Online. A new dialog will prompt for a License ID and Password (both are required). Specifying an installation name is optional, but helpful when performing multiple installations for different machines and/or users.



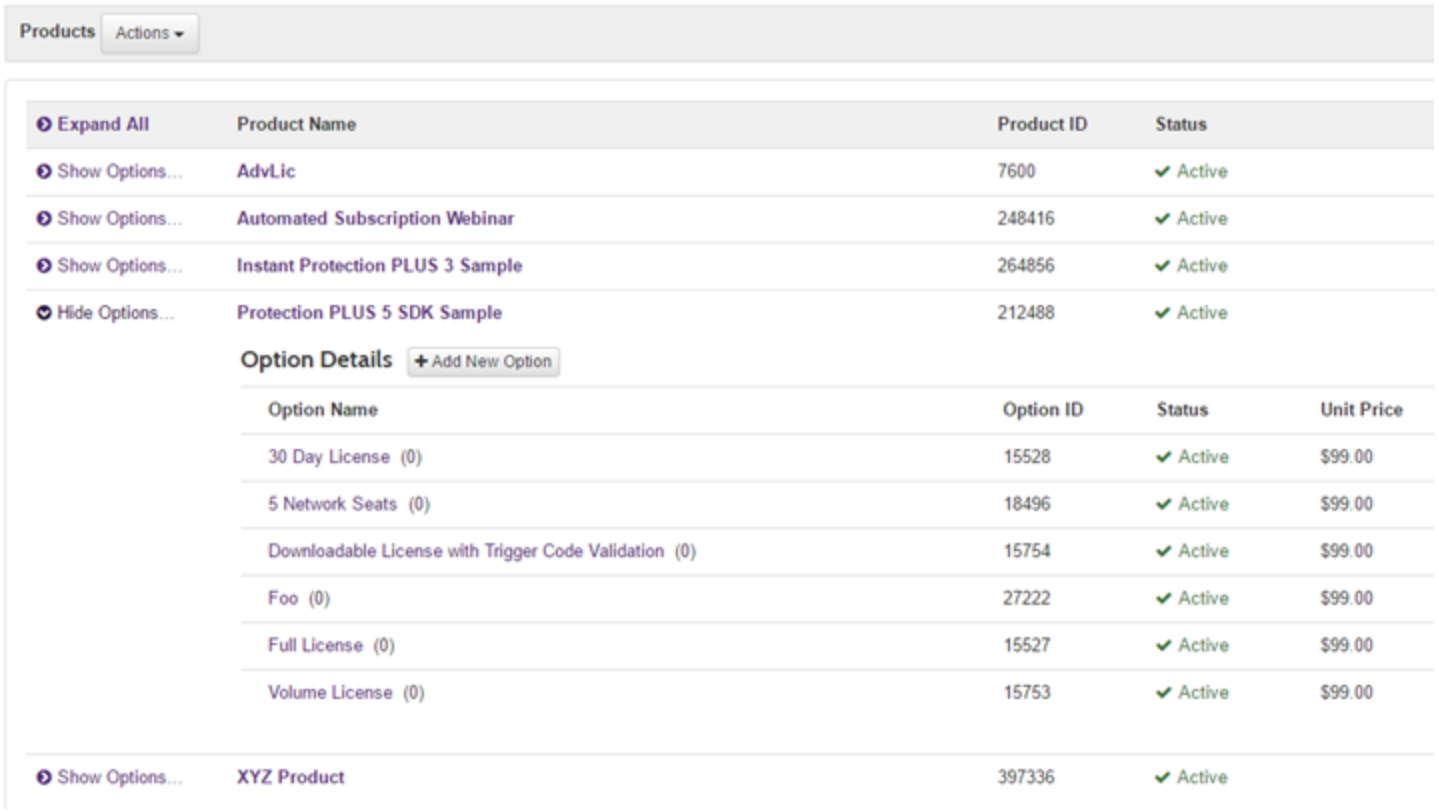
For ease of use, all of the Protection PLUS 5 SDK sample applications use a generic Test Author account with its own *Encryption Key ID* . We will use this Test Author account on Instant SOLO Server to generate the License ID and password necessary to activate the Licensing Sample. To log into Instant SOLO Server, visit **https://secure.softwarekey.com/solo/authors/Default.aspx**. Use the Test Author credentials given below:

- User ID: test
- Password: test

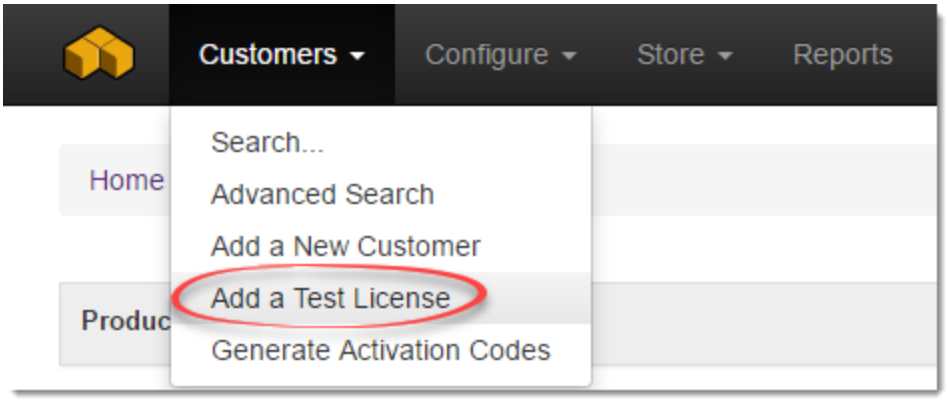


Once logged in, go to the menu *Configure / Products*.

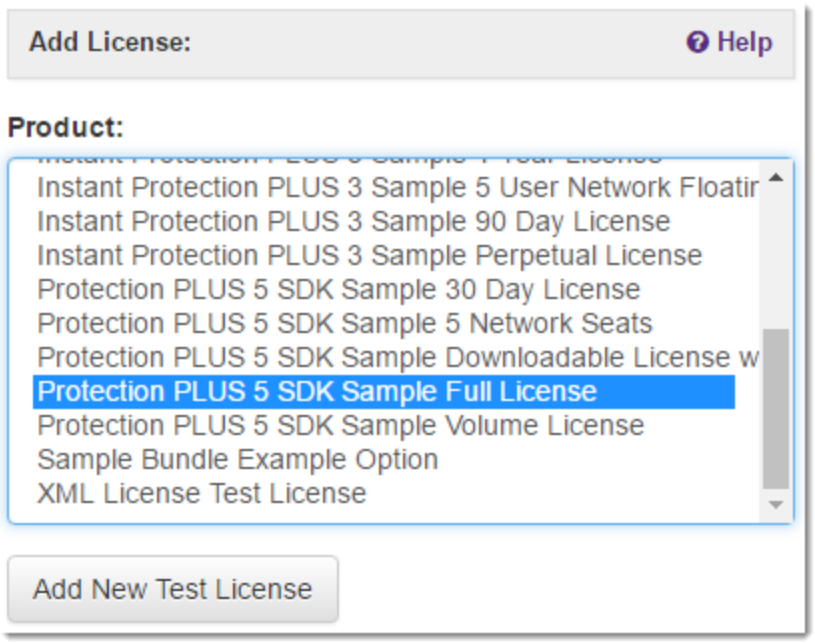
The Product List page will show all of the products available to the Test Author. Expand Show Options beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for activating the Licensing Sample application. For this tutorial, we will generate a license for the Full License.



To add a test license for this product option, mouse over Customers in the navigation bar, and click "Add Test License".



Select the Protection PLUS 5 SDK Sample Full License and click Test License.



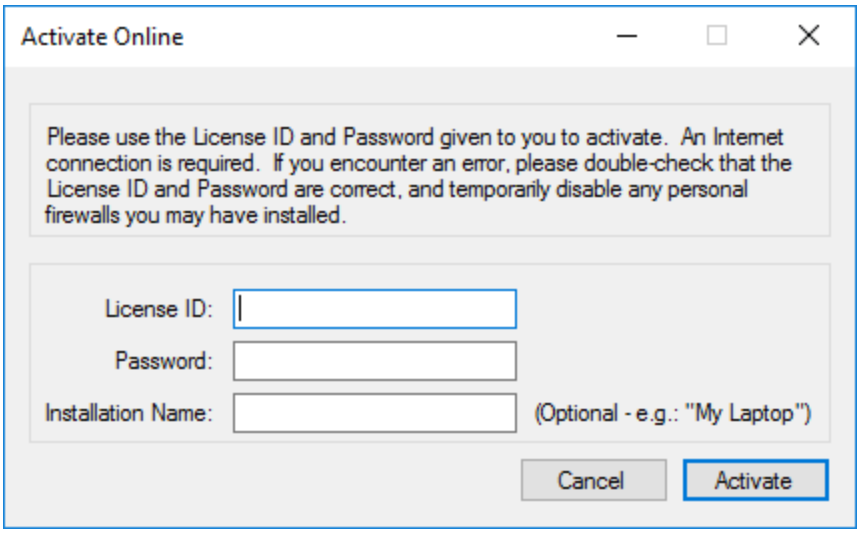
Press OK to continue when prompted.

Important:

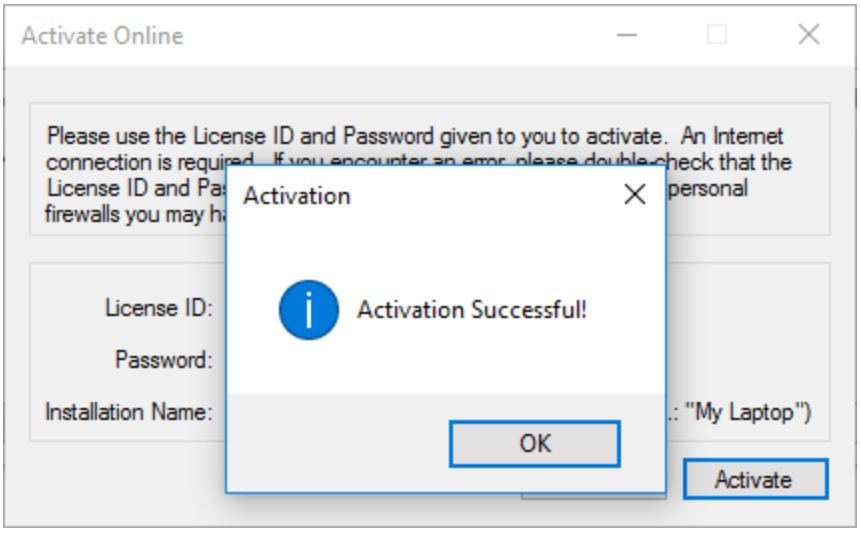
Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.

The Test License has now been created. We now have a License ID and Password to continue testing the Licensing Sample.

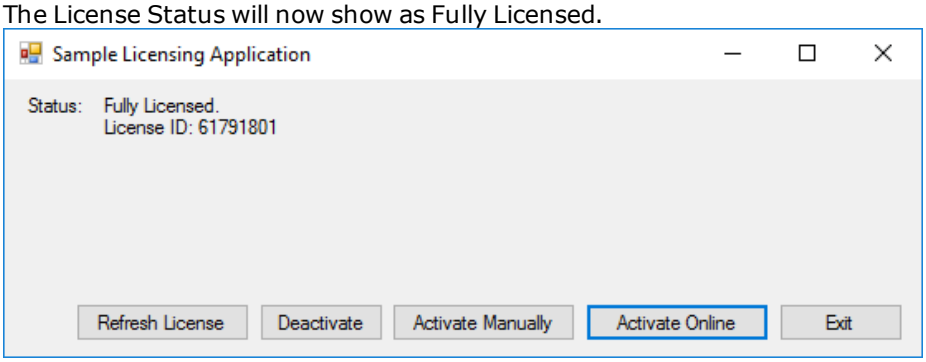
Return to the activation prompt for our Licensing Sample. Enter the License ID and Password in their respective fields and, optionally, an Installation Name. Click Activate to communicate with Instant SOLO Server for license validation.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



The License Status will now show as Fully Licensed.

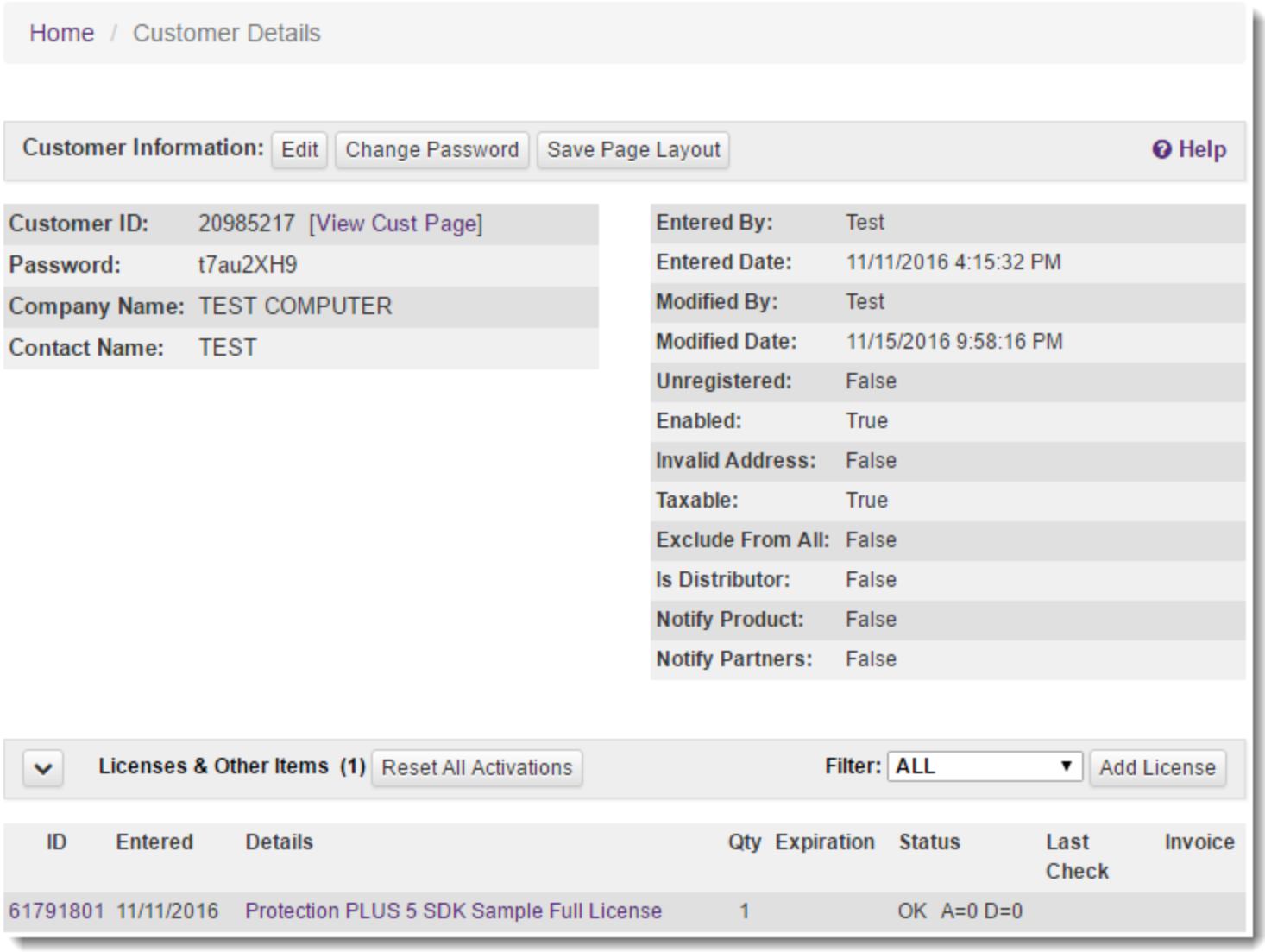


Step 3- Refresh a License

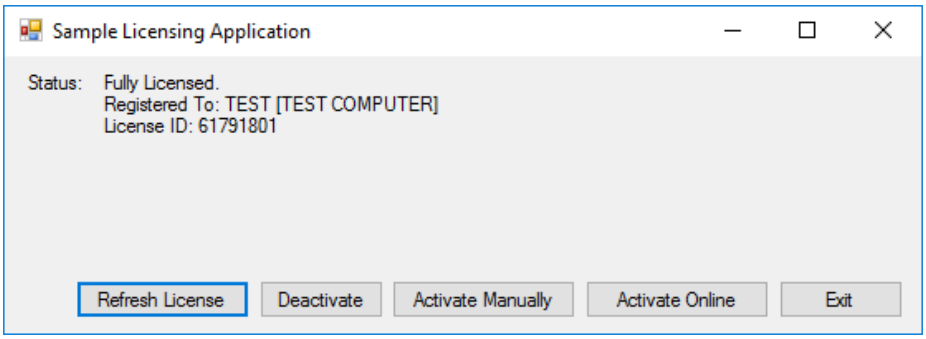
A license can be refreshed with updated information sent fromInstant SOLO Server. We can modify customer details from the License Details page on Instant SOLO Server. On the License Details page, click Edit, located to the right of the Customer ID.

Modify the Company Name and additional fields as you please. Uncheck the check box setting to the right of Unregistered. Scroll to the bottom and click Save.

After the Customer Details have been saved, the page returns to the Customer Information page. This page includes customer details we have edited in addition to licenses and previous orders.

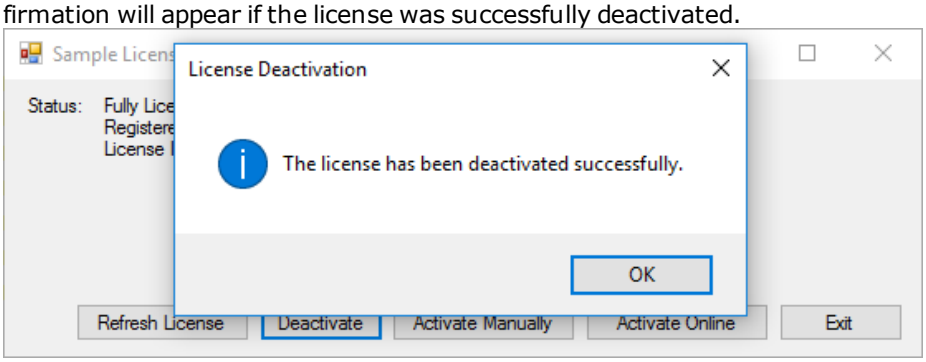


Return to the Licensing Sample Main Dialog to proceed with a license refresh. Click Refresh License. If the licensed successfully refreshed, the License Status will now update with the modified customer details.

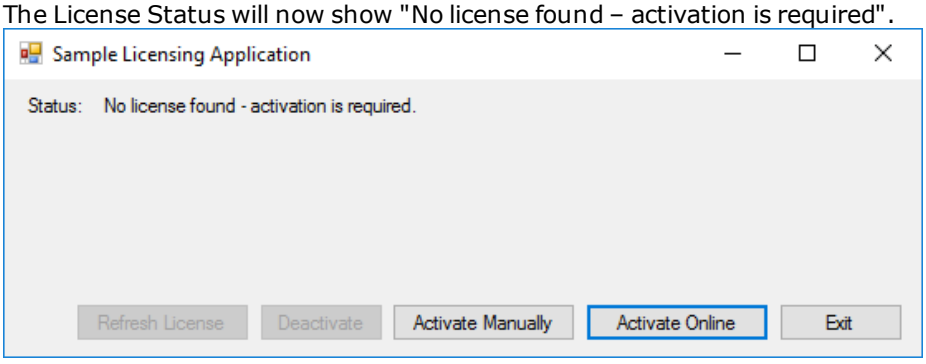


Step 4 - Process a Manual Activation

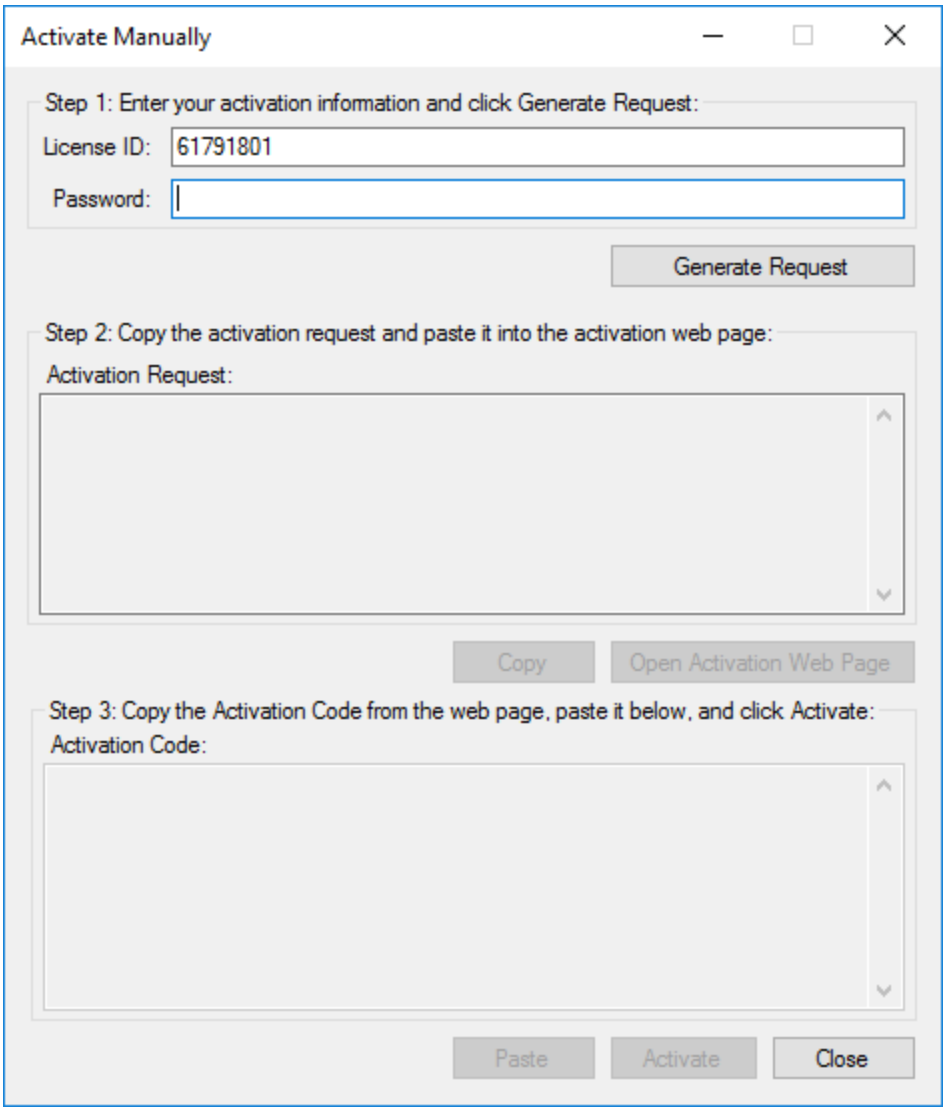
If you previously activated the Licensing Sample online using Instant SOLO Server, you will need to deactivate the license before testing manual activations. In the Main Dialog of the Licensing Sample, click Deactivate. A confirmation will appear if the license was successfully deactivated.



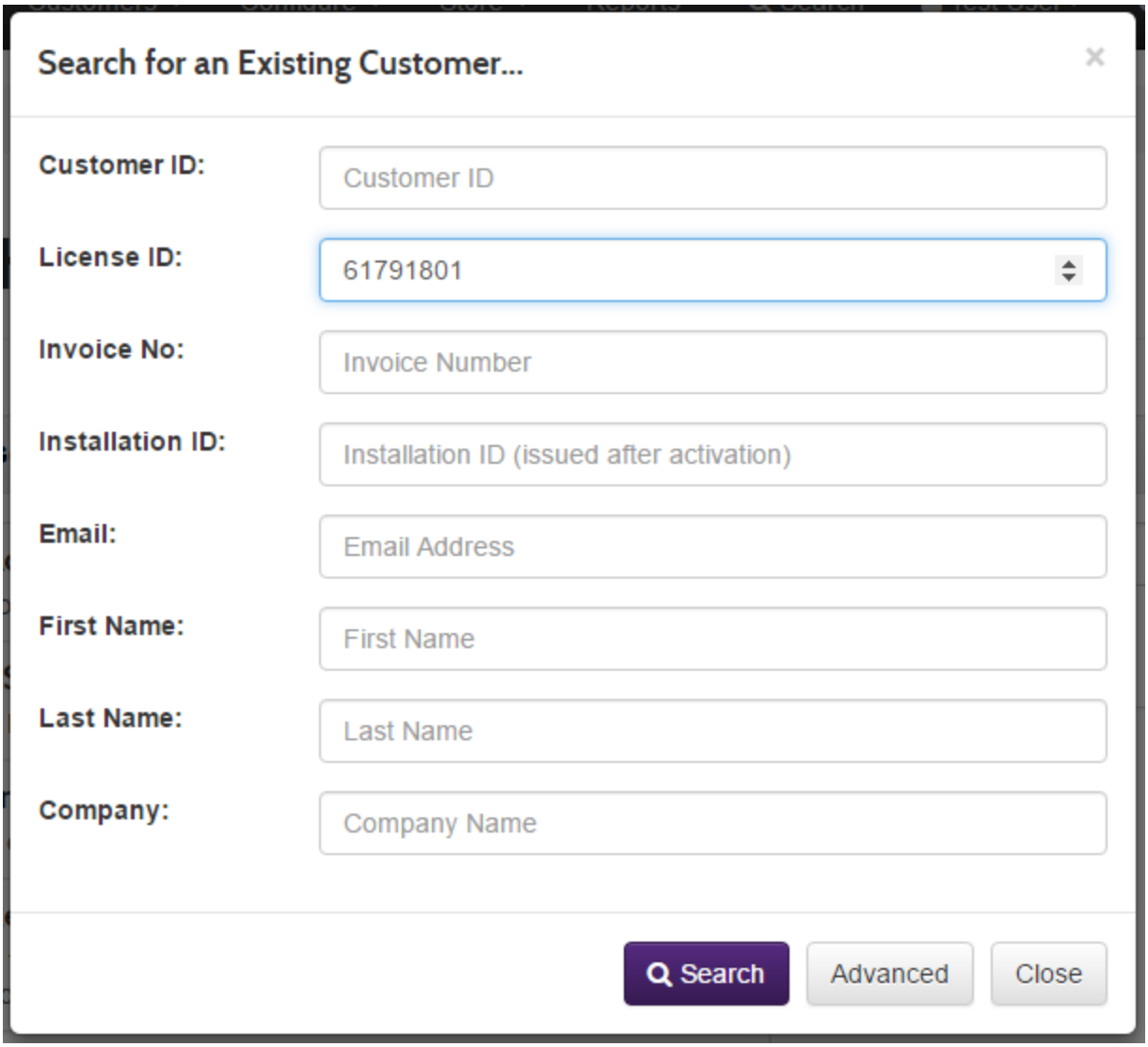
The License Status will now show "No license found – activation is required".



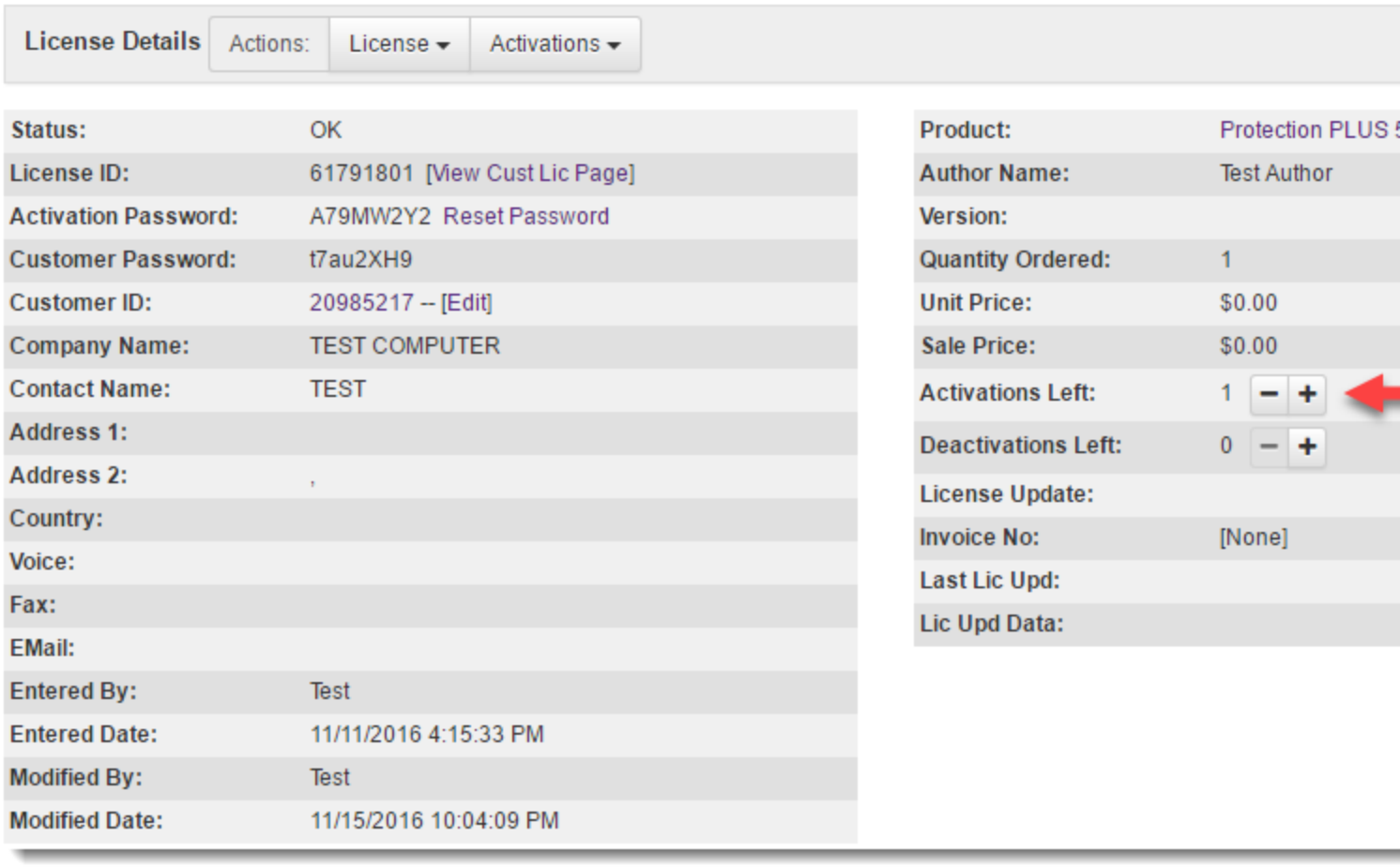
Click Activate Manually to proceed with testing a manual activation. A manual activation allows for activations from another computer or by e-mail. Follow the instructions in Step # 2 to generate a new license. Alternatively, we can also check the number of activations remaining for the License ID generated when we tested automatic activations online using Instant SOLO Server.



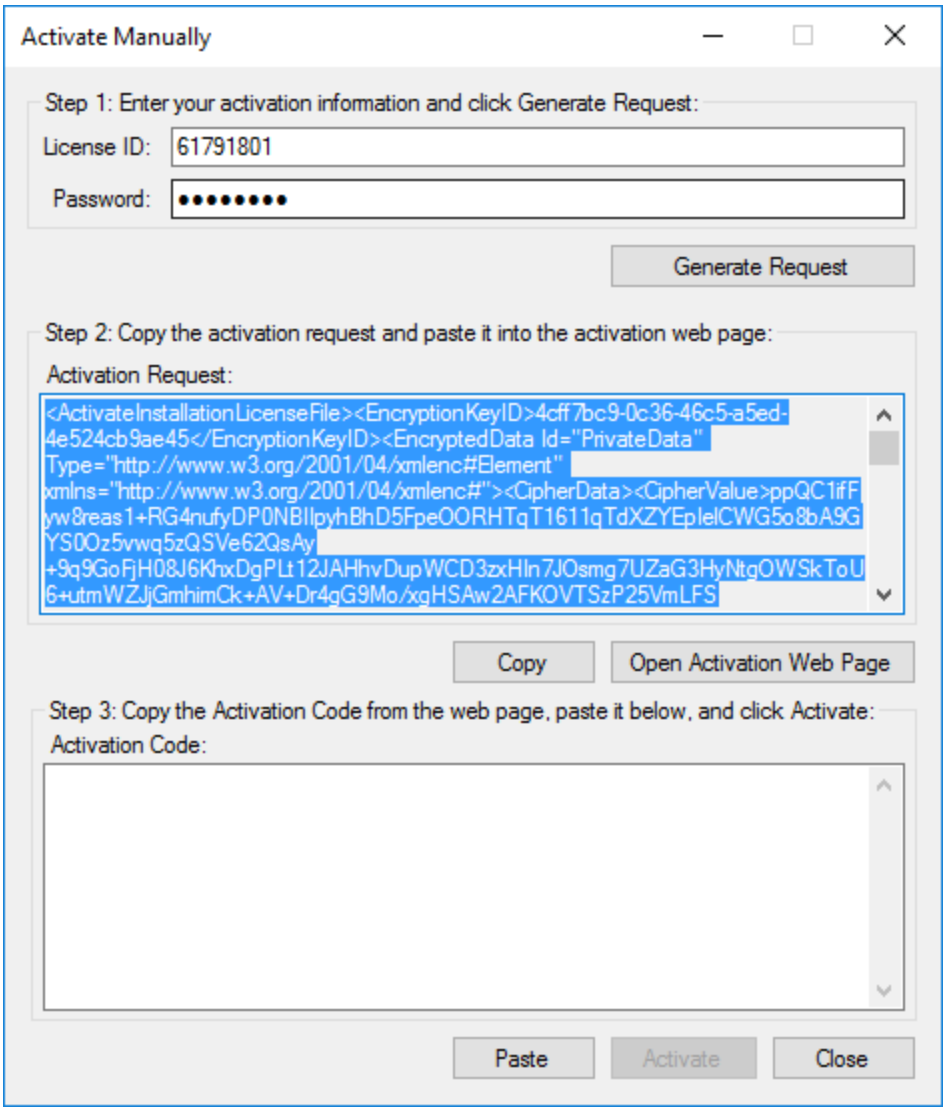
Return to Instant SOLO Server using the Test Author account. Use the Search menu to perform a quick search for the License ID previously used to activate online (If you no longer have this number, proceed to create a new License ID as shown in Step # 2).



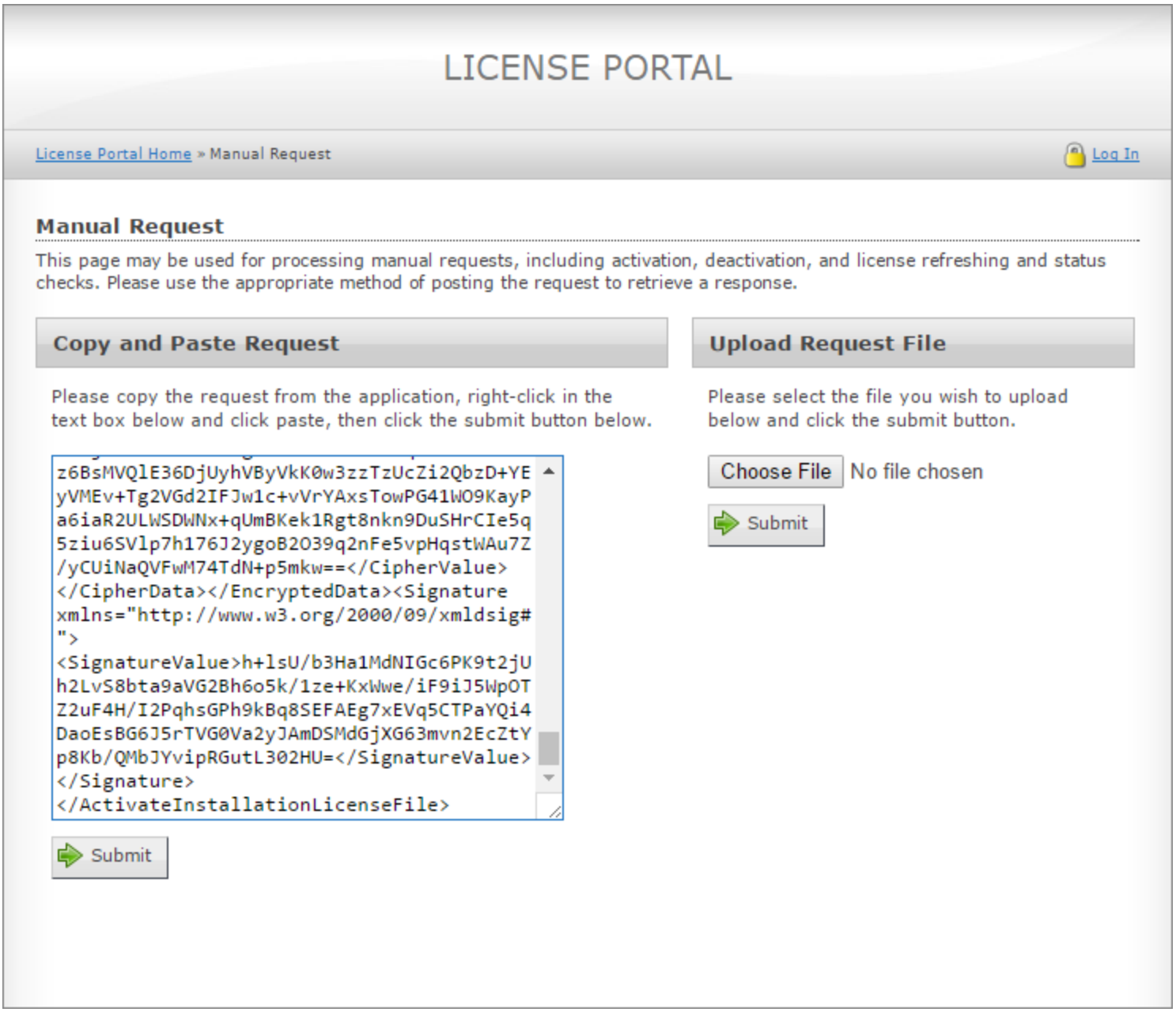
When we clicked on deactivate license, Instant SOLO Server automatically incremented the amount of activations left by 1. If you need to manually increase the number of activations left, you can increment this number by clicking the + button to the right of the Activations Left field.



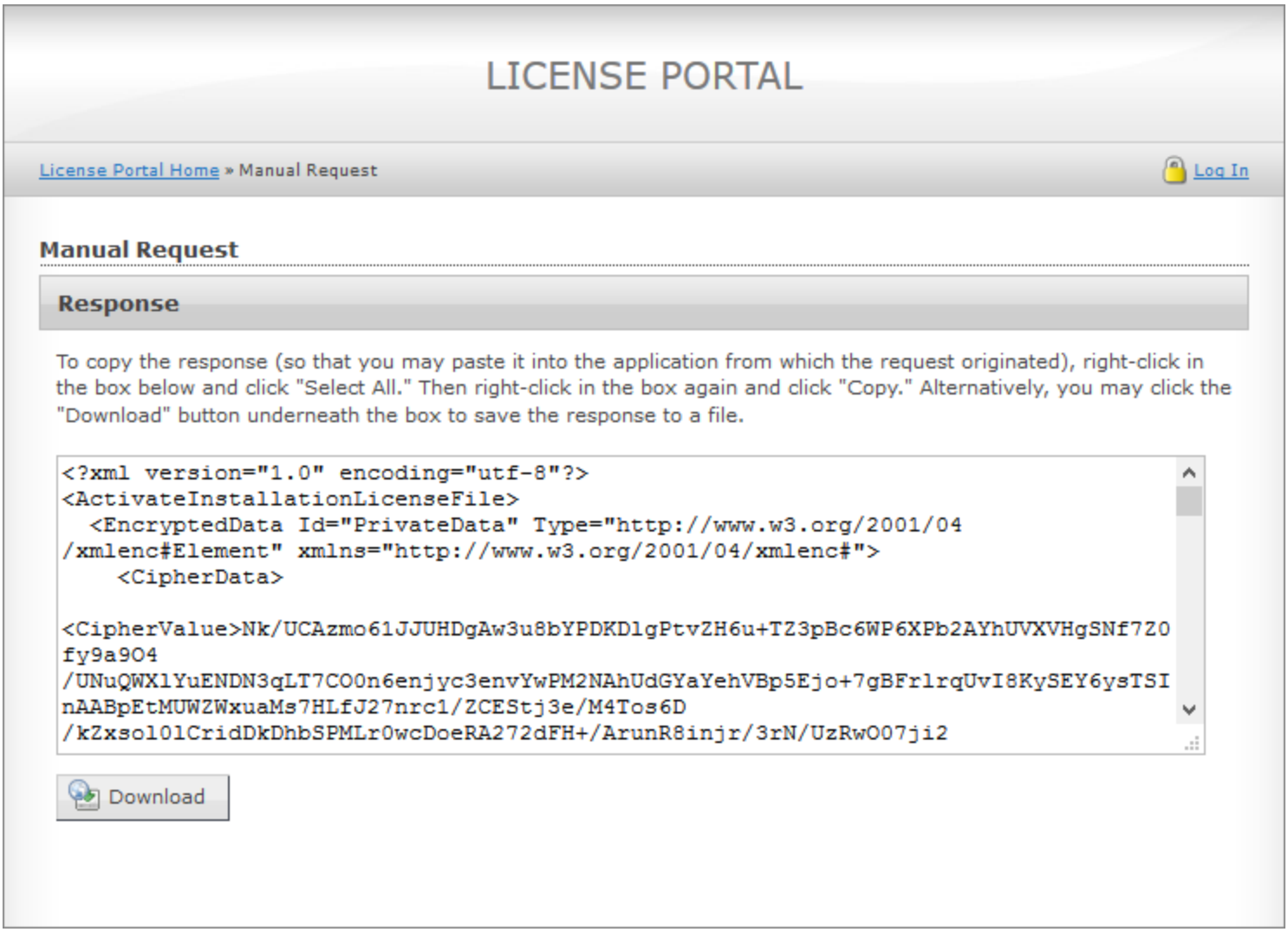
Return to the Activate Manually for our Licensing Sample. Enter the License ID and Password in their respective fields and click Generate Request.



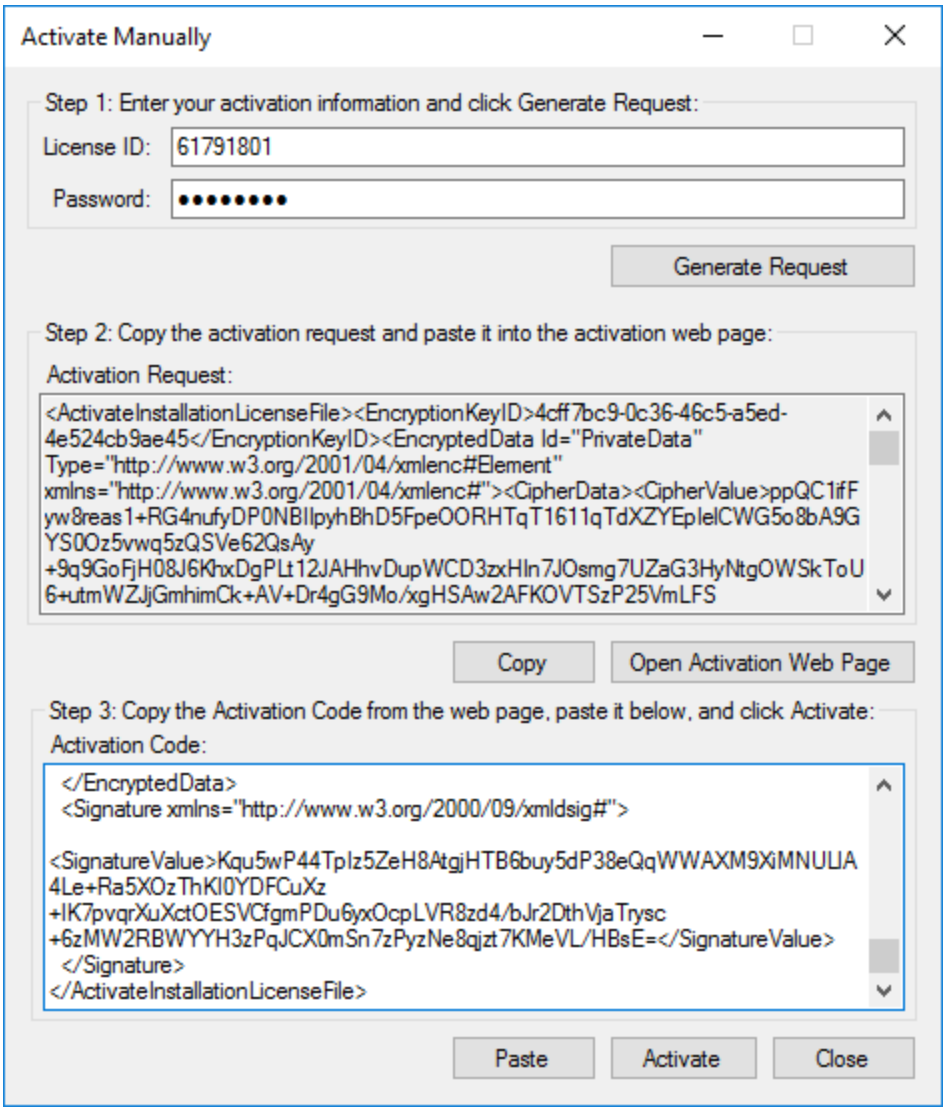
This dialog will generate an encrypted activation request to be processed by the activation server. Click Copy to copy this data to the clipboard. Next, click Open Activation Web Page. This web page is accessible from another computer with access to the Internet in the event that an offline workstation needed to be activated. Once you arrive to the License Portal page, paste the encrypted activation request data, and click Submit.



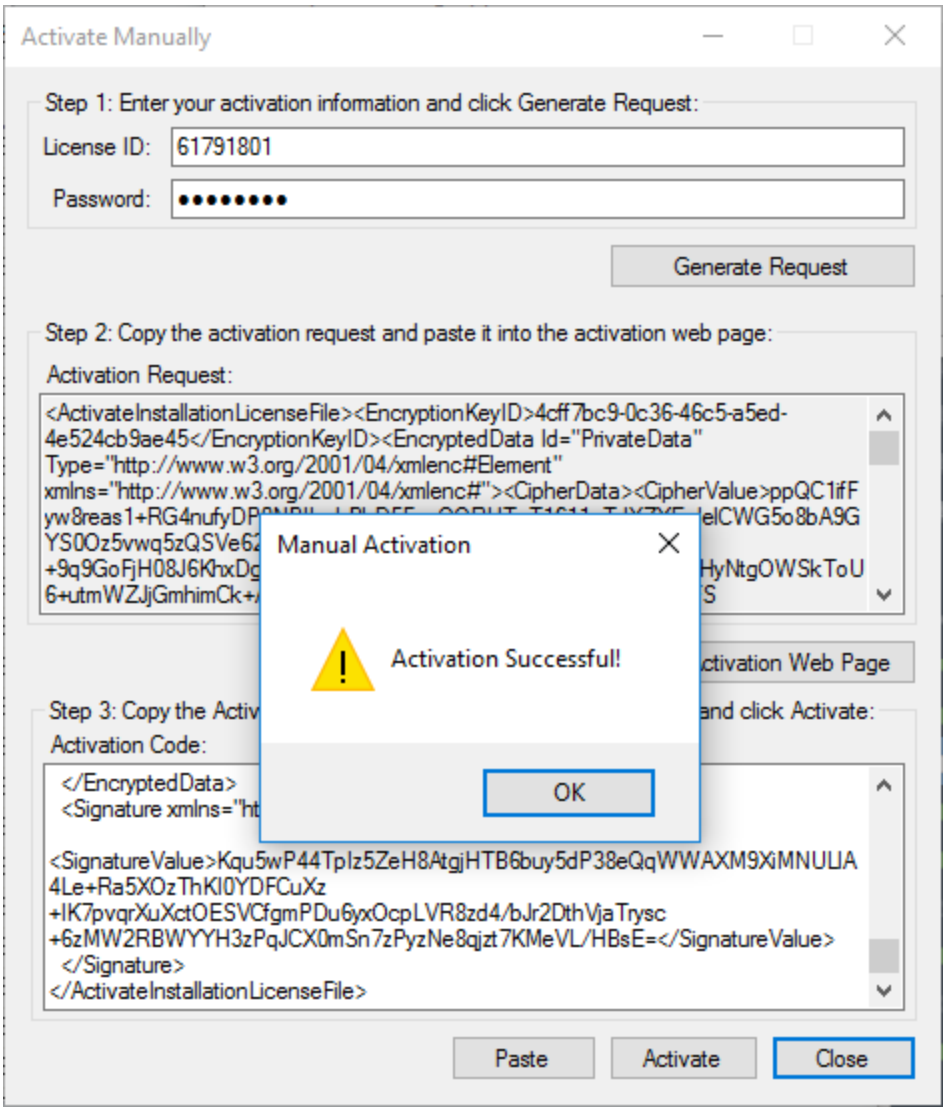
The License Portal will now generate an encrypted response. Copy this data to later paste in the Activation Code field of the activation window.



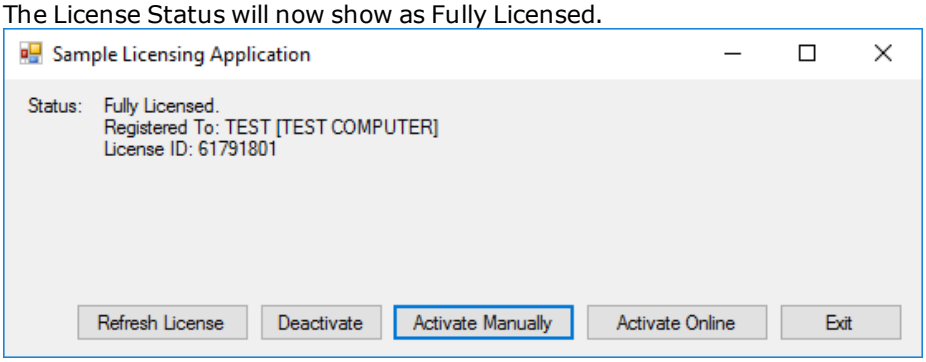
Return to the activation window and paste the encrypted response into the Activation Code field and click Activate.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



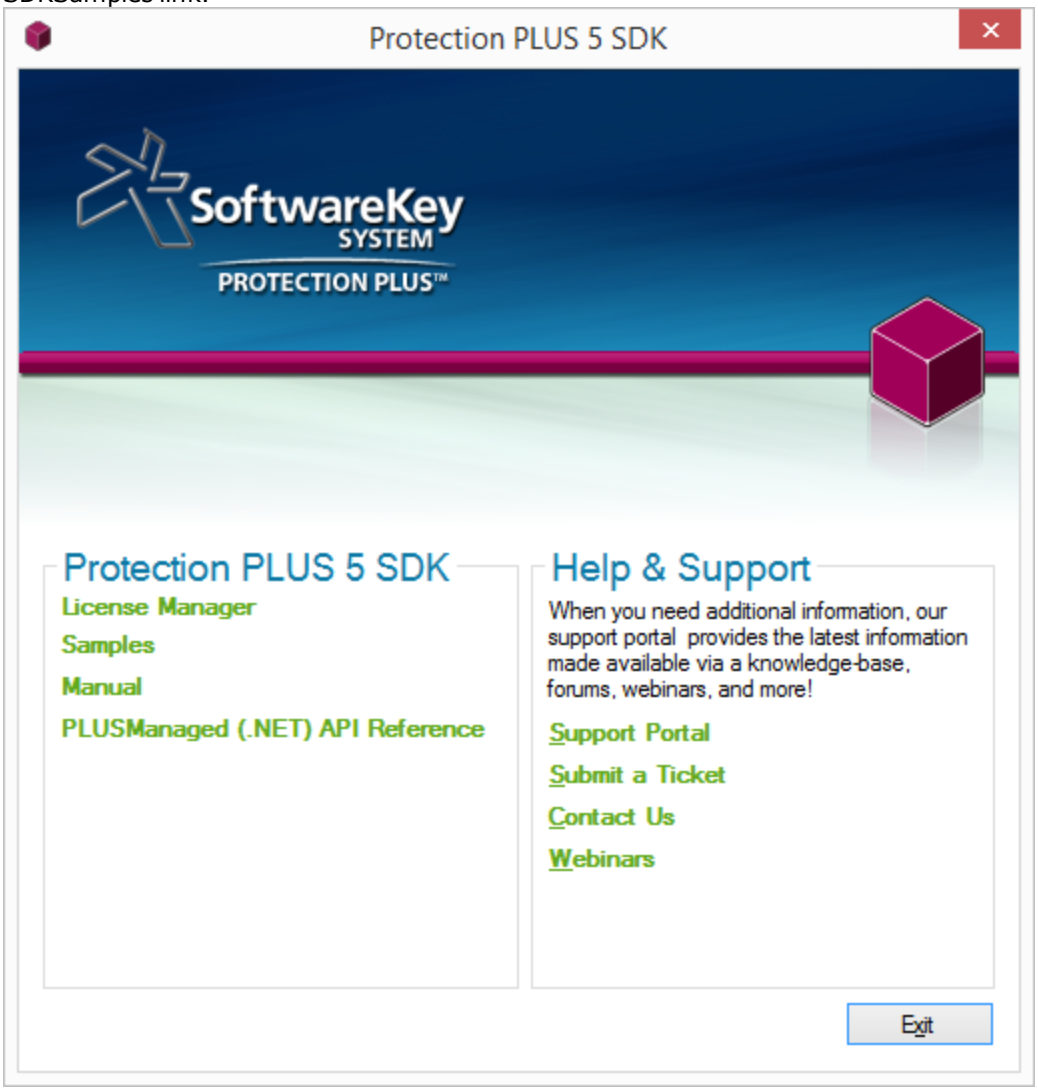
The License Status will now show as Fully Licensed.



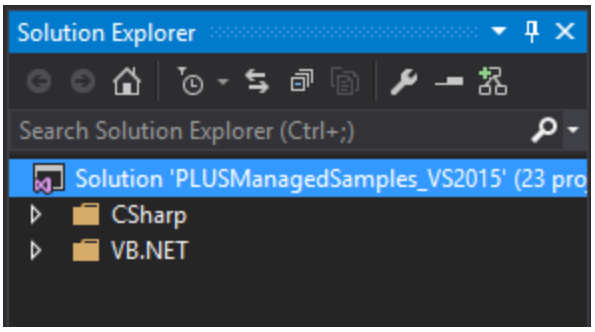
PLUSManaged Simple Text Editor Sample

Step 1 – Opening the SimpleTextEditor Sample Project

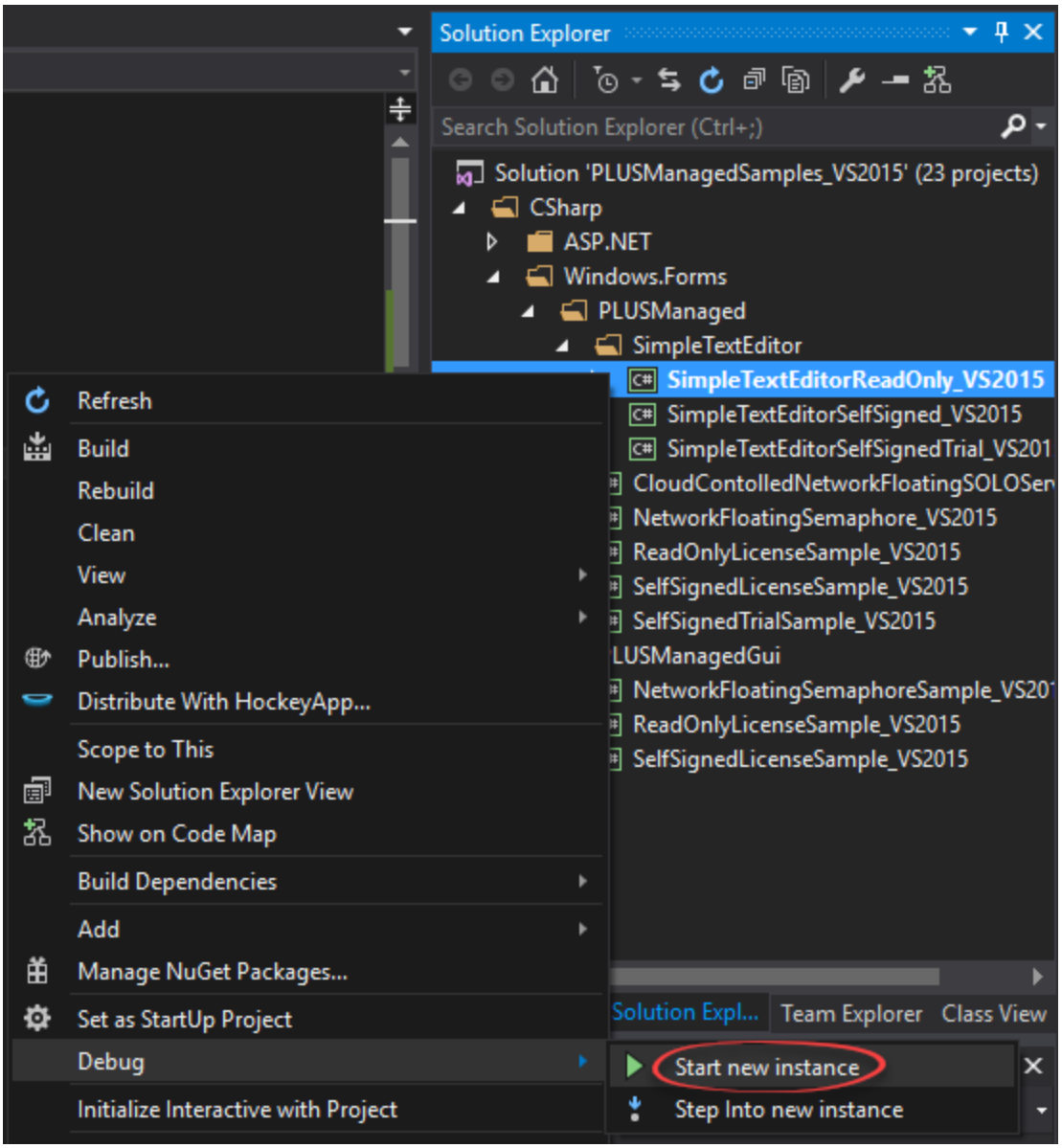
Open the PLUSManagedSamples *.sln file corresponding to your version of Visual Studio. The Samples directory can be opened by running Protection PLUS 5 SDK from your start menu/screen, and clicking on the Protection PLUS 5 SDKSamples link.



Once the project has loaded in Visual Studio, expand the programming language you wish to proceed with (C# or VB.NET).

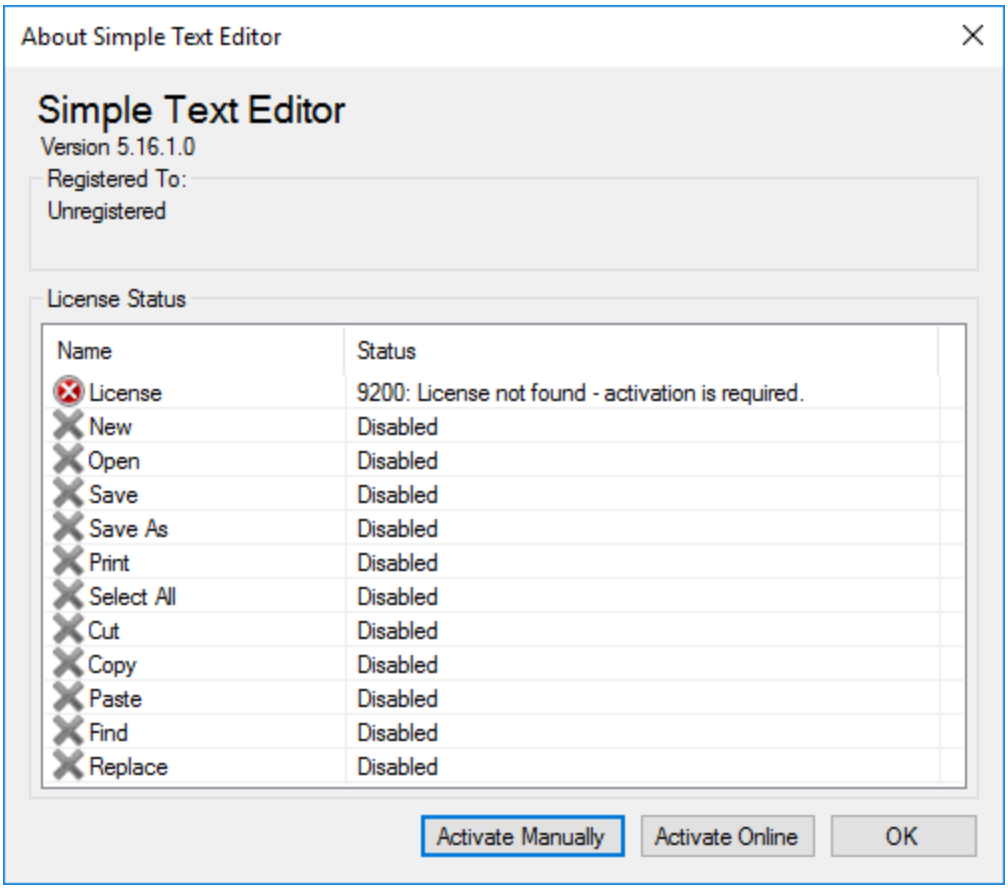


Expand the Windows.Forms folder and PLUSManaged folder. Locate the SimpleTextEditorReadOnly Sample. Right-click this sample and select Debug > Start New Instance.



Step 2- Activating automatically with Instant SOLO Server

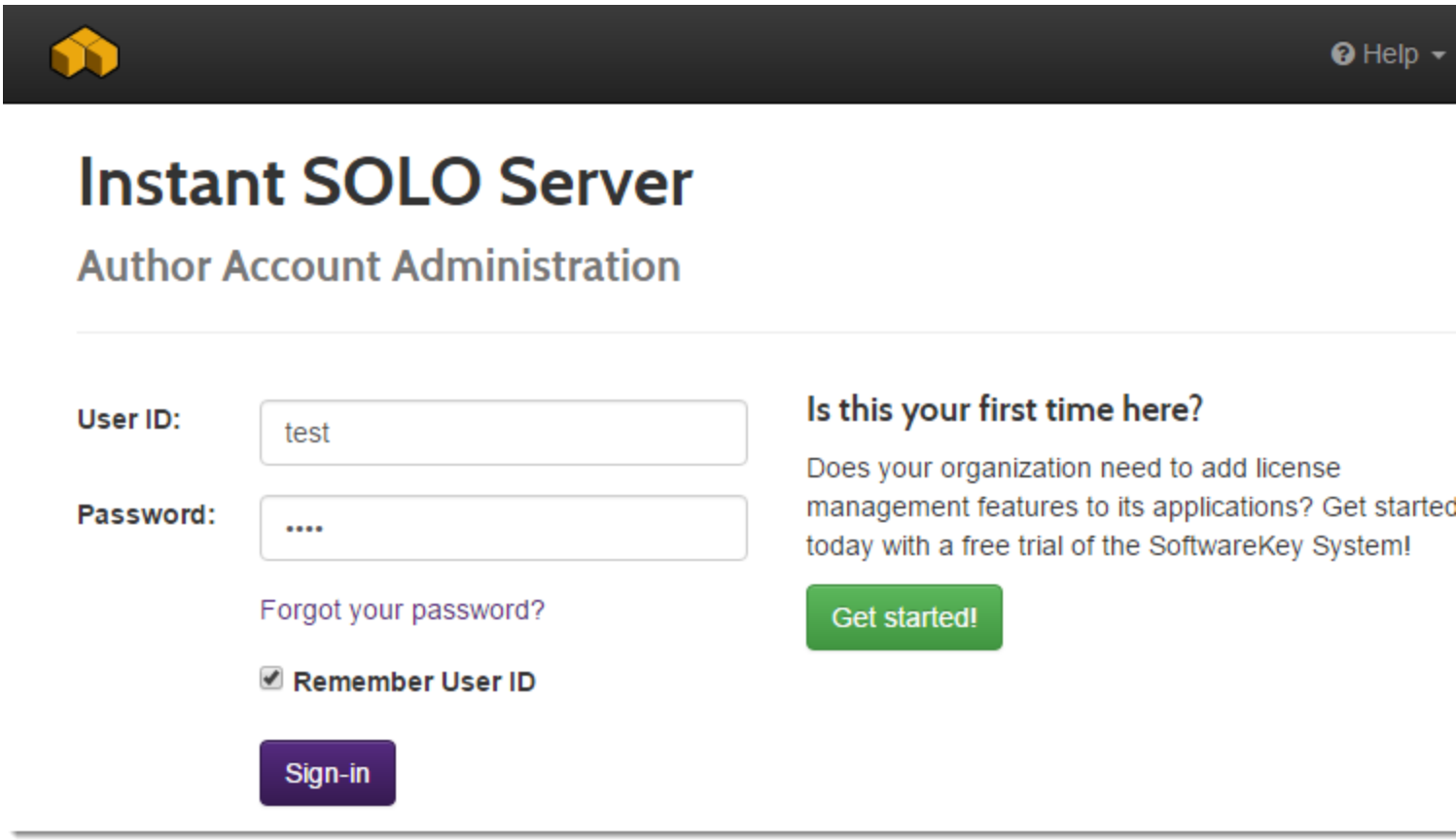
Upon launching, the Simple Text Editor will show the current license status along with the features that are enabled or disabled. To activate the application automatically using SOLO Server, click on Activate Online. A new dialog will prompt for a License ID and Password (both are required). Specifying an installation name is optional, but helpful when performing multiple installations for different machines and/or users.



For ease of use, all of the Protection PLUS 5 SDK sample applications use a generic Test Author account with its own Encryption Key ID. We will use this Test Author account on SOLO Server to generate the License ID and password necessary to activate the Simple Text Editor Sample.

To log into Instant SOLO Server, visit **https://secure.softwarekey.com/solo/authors/Default.aspx**. Use the Test Author credentials given below:

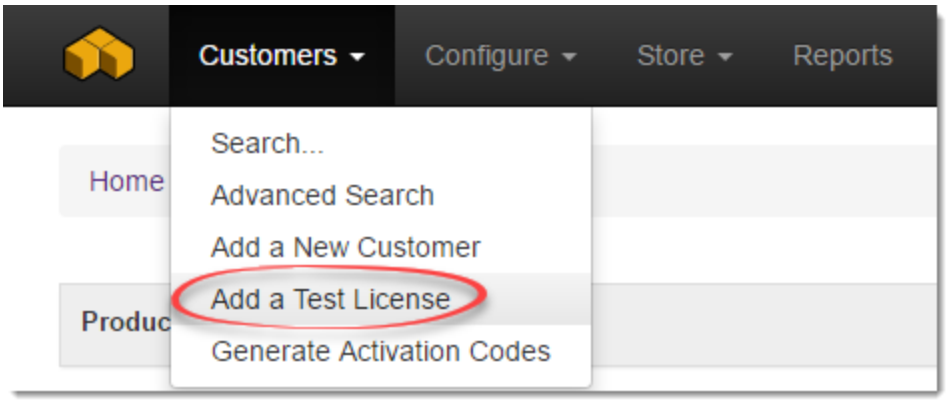
Login ID: test
Password: test



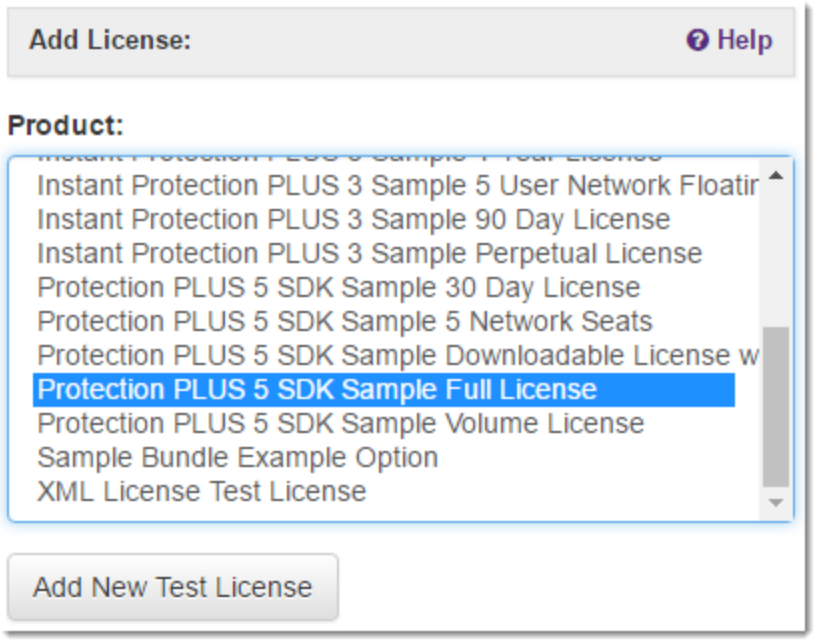
Once logged in, Go to the menu *Configure / Products*.

The Product List page will show all of the products available to the Test Author. Expand "Show Options" beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for activating the Simple Text Editor Sample application. For this tutorial, we will generate a license for the Full License.

To add a test license for this product option, mouse over Customers in the navigation bar, and click "Add Test License".

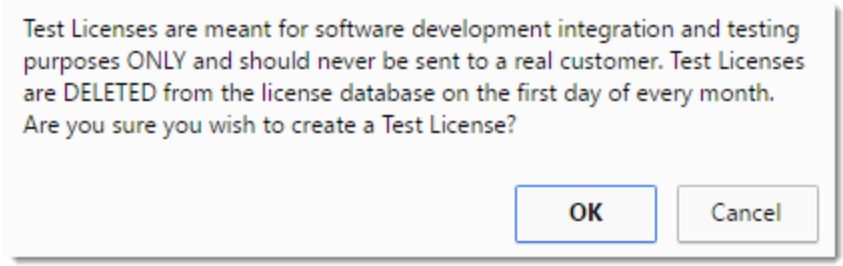


Select the Protection PLUS 5 SDK Sample Full License and click Add New Test License.



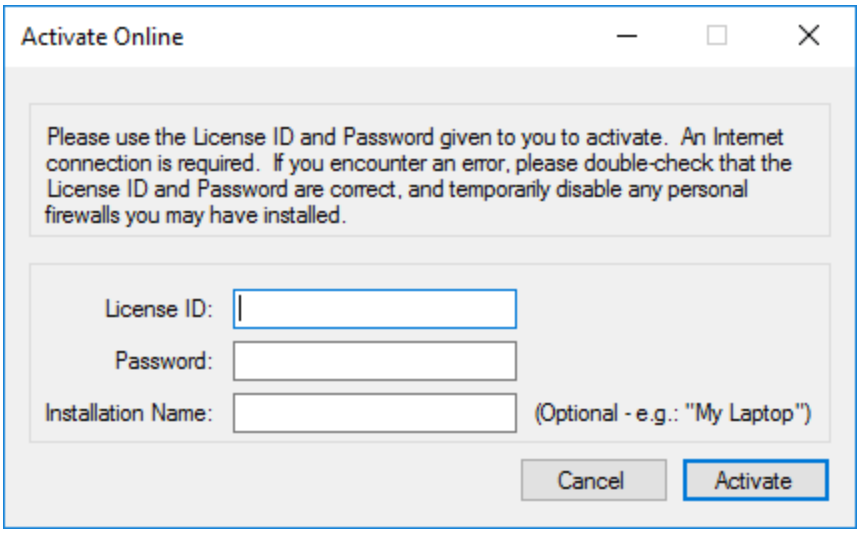
Important:

Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.

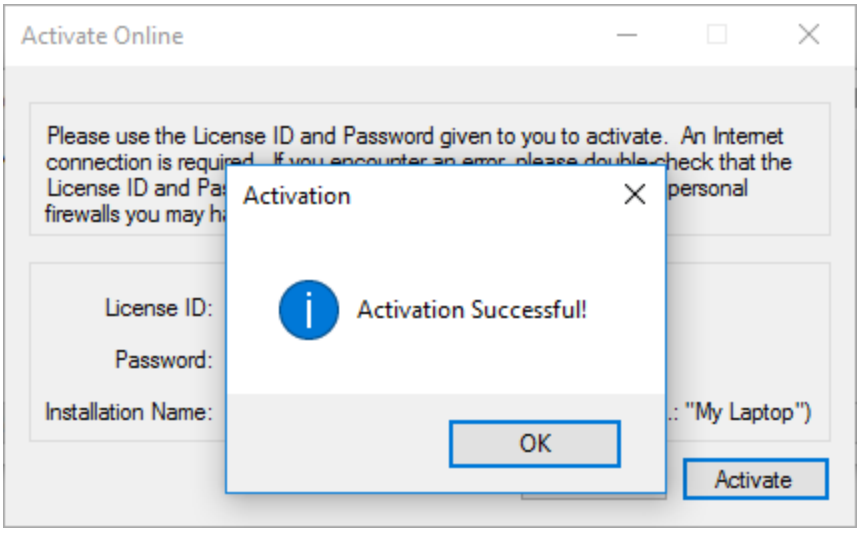


The Test License has now been created. We now have a License ID and Password to continue testing the Simple Text Editor Sample.

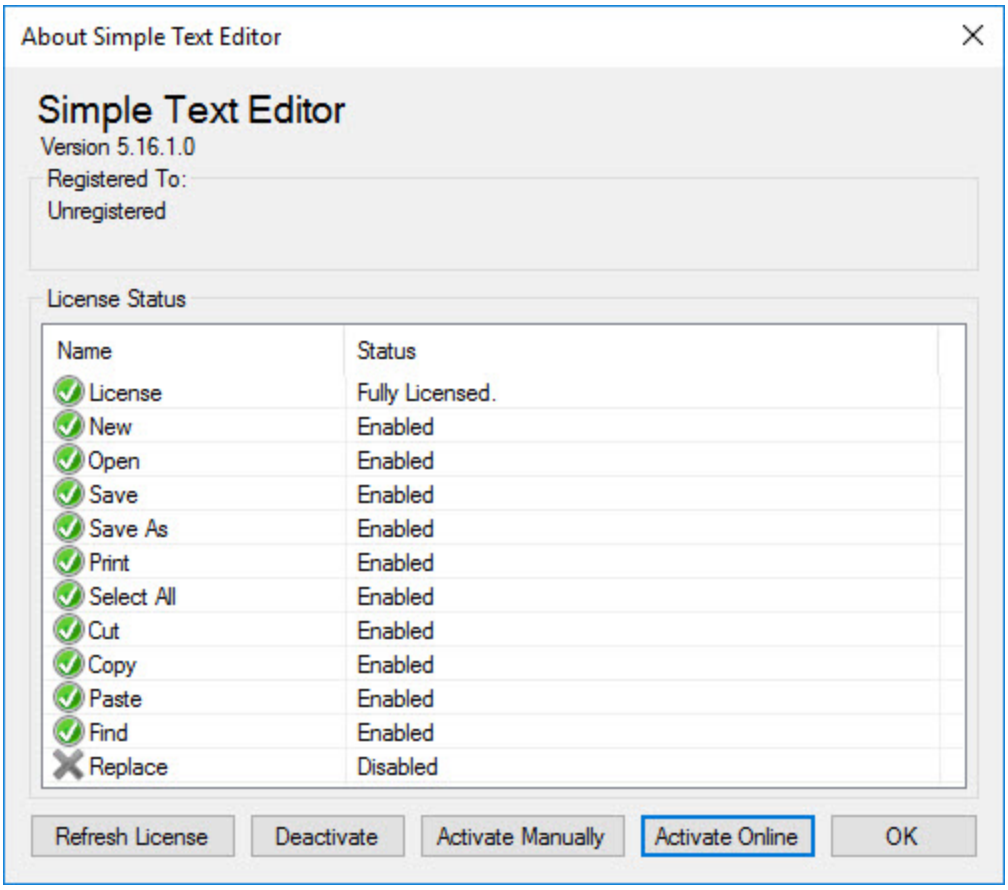
Return to the activation prompt for our Simple Text Editor Sample. Enter the License ID and Password in their respective fields and, optionally, an Installation Name. Click Activate to communicate with SOLO Server for license validation.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



The License Status will now show as Fully Licensed.

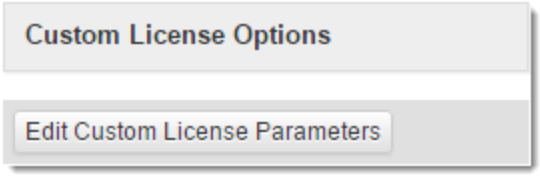


Step 3- Refresh a License

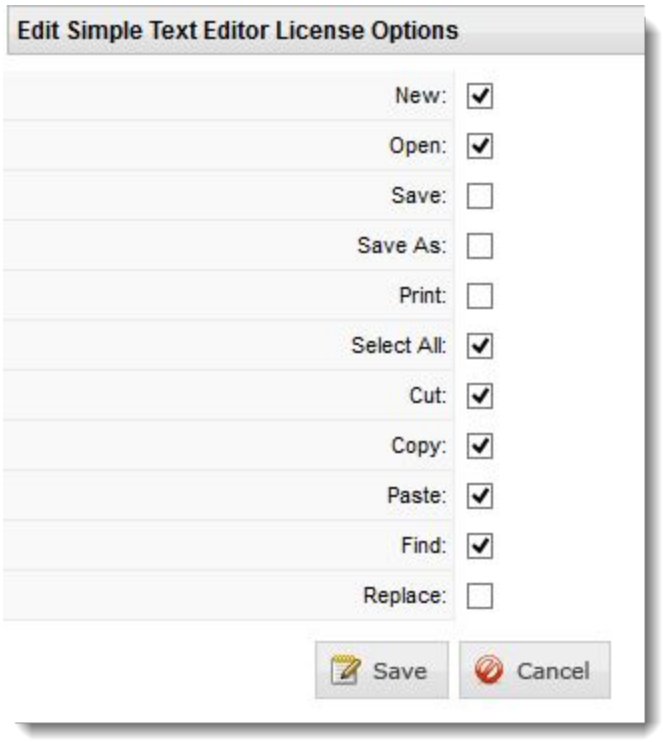
A license can be refreshed with updated information sent from Instant SOLO Server. We can update the end-user's name and address, or limit an application's features by editing custom license parameters. The SimpleTextEditor samples also show you how you can enable and disable individual features for licenses. In this case, these samples conditionally enable/disable menu and toolbar buttons based on what bits are enabled on the TriggerCodeFixedValue property (set in the Product Option in Instant SOLO Server) or the UserDefinedNumber1 property (set in the user defined fields for the License ID in Instant SOLO Server).

For this example, we create a custom license option button. A similar customization can be done using our professional services. Contact technical support for more details.

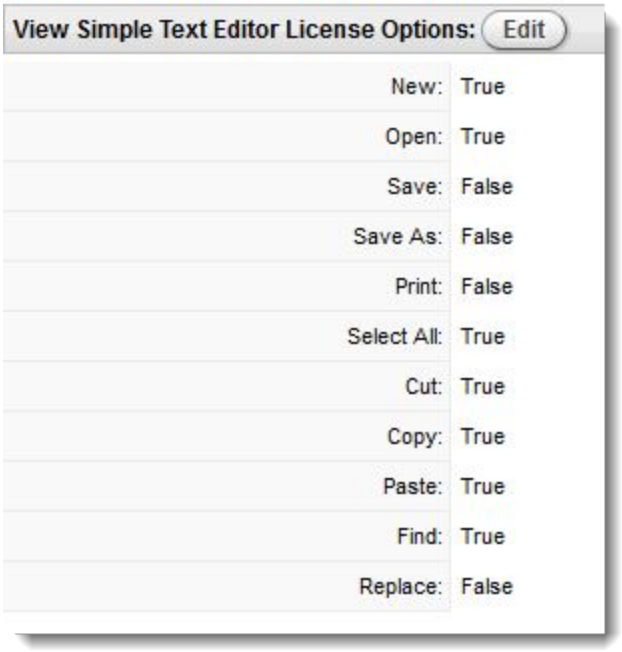
On the License Details page, locate the License Options and click on Edit Custom License Parameters.



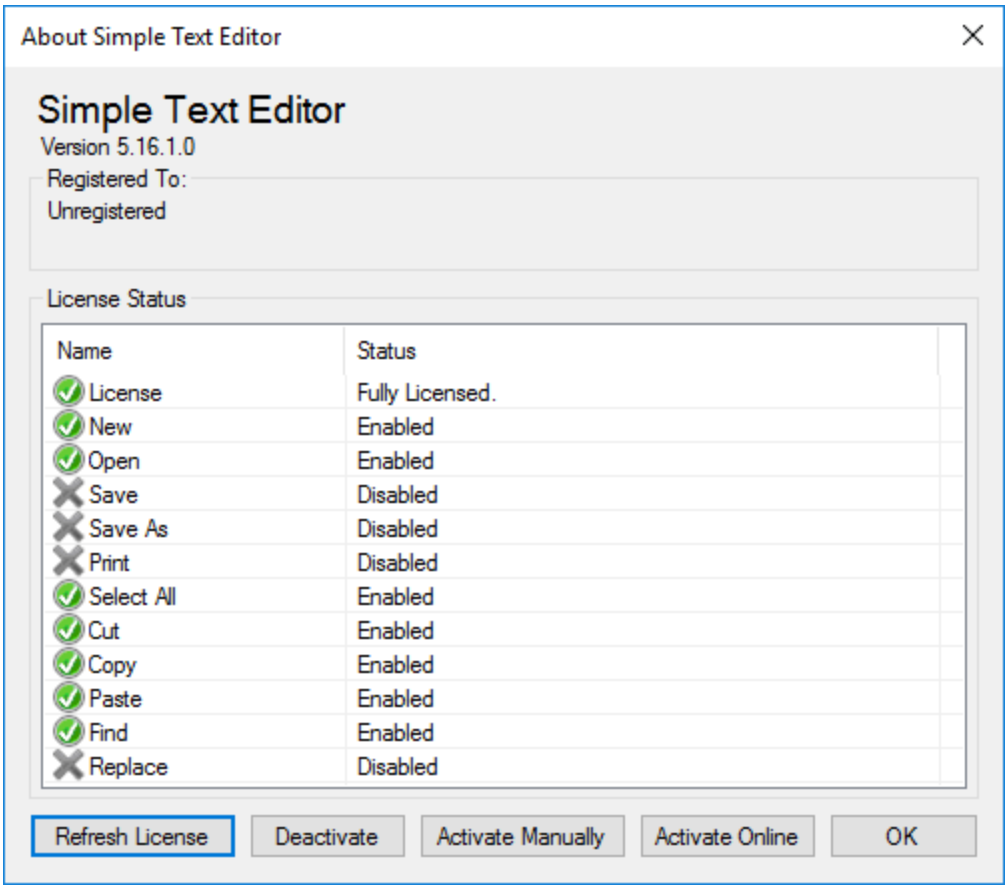
You will notice that the features that are currently enabled. To disable certain features, uncheck the corresponding box beside the feature you wish to disable, and click Save.



The disable features will now show false under the Simple Text Editor License Options.

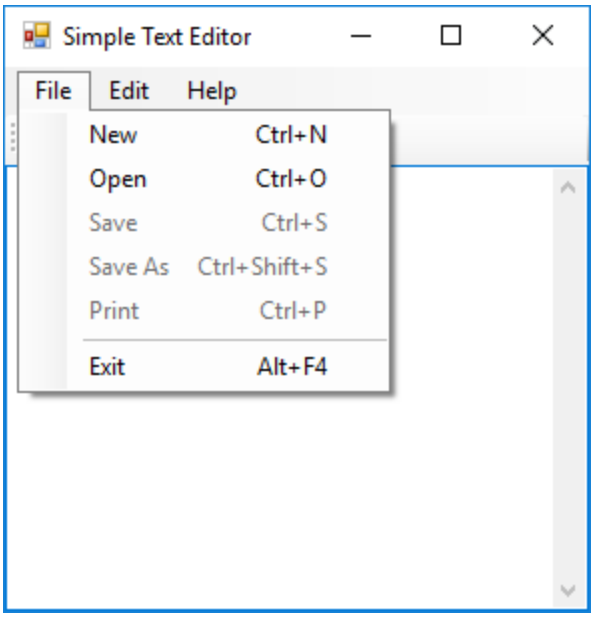


Return to the License Status Dialog for the Simple Text Editor and click Refresh License. When the license successfully refreshes, the disabled features will show their updated status.



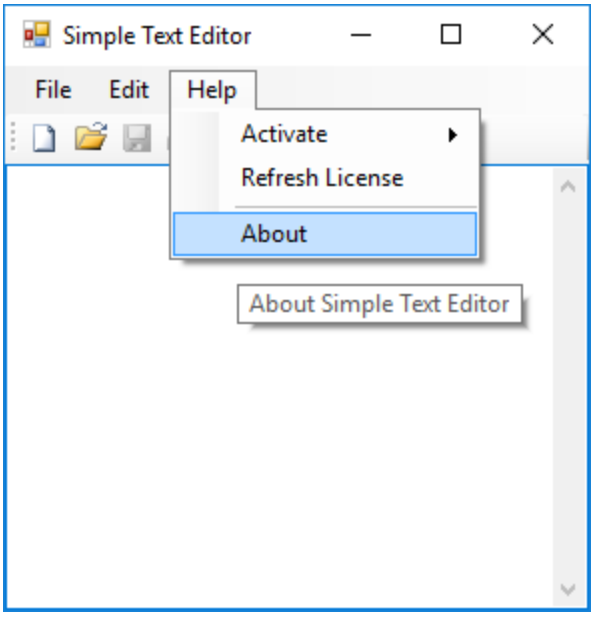
Click OK to now show the Simple Text Editor. Notice that in the File Menu, the features we disabled are now disabled in the application. The appropriate icons have also been disabled.

>

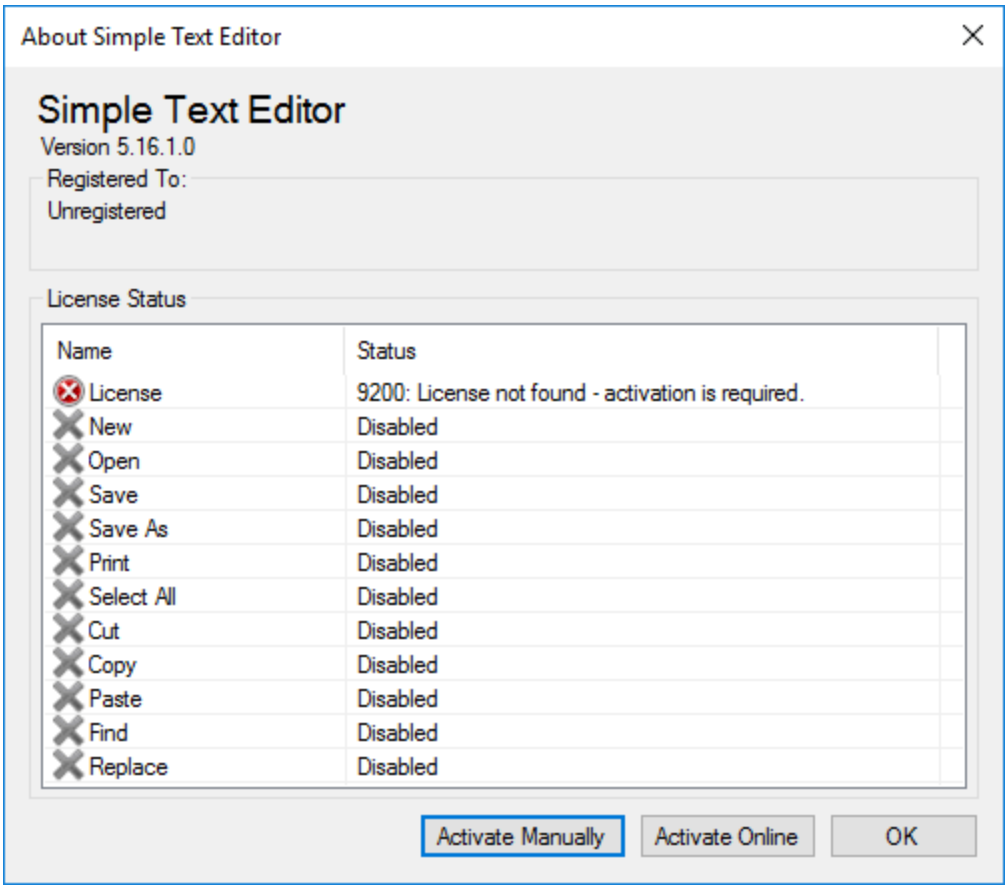


Step 4- Process a Manual Activation

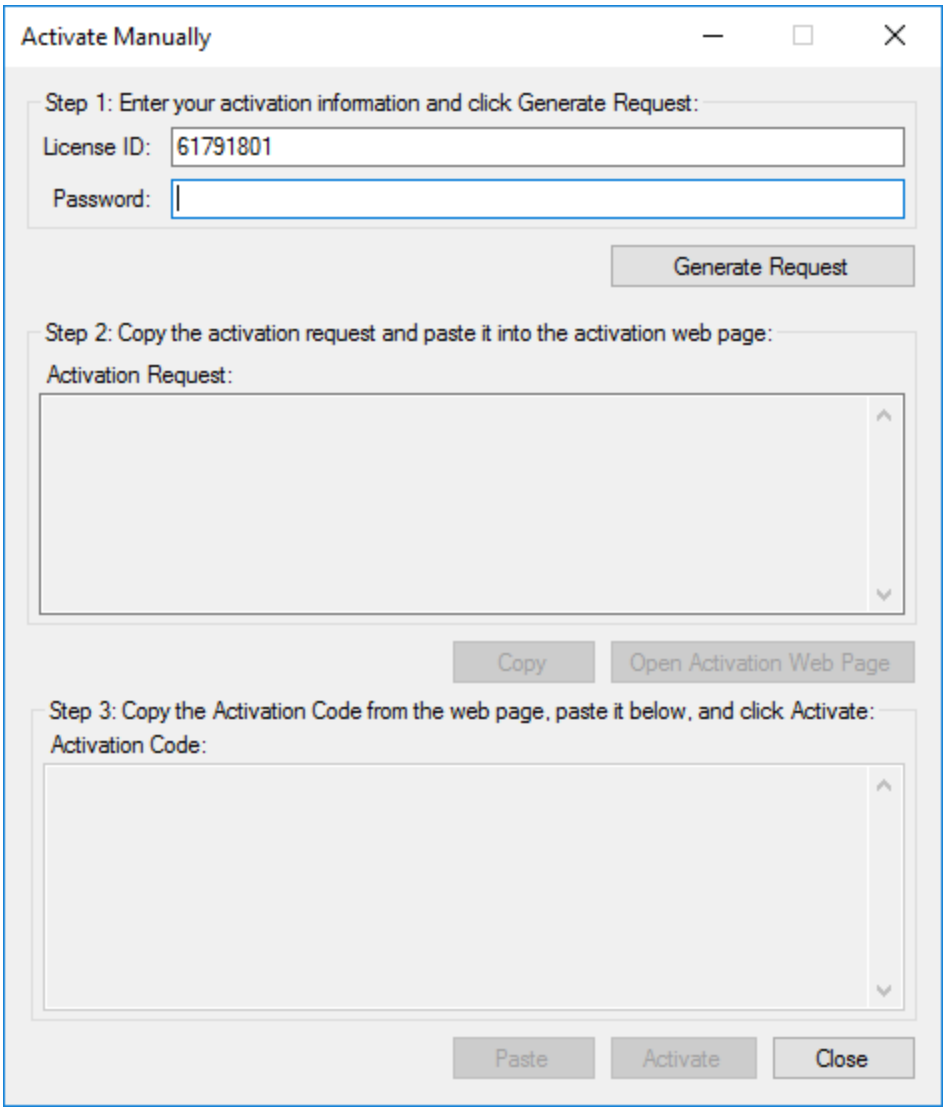
If you previously activated the Simple Text Editor online using Instant SOLO Server, you will need to deactivate the license before testing manual activations. To bring up the License Status dialog of the Simple Text Editor, click Help > About.



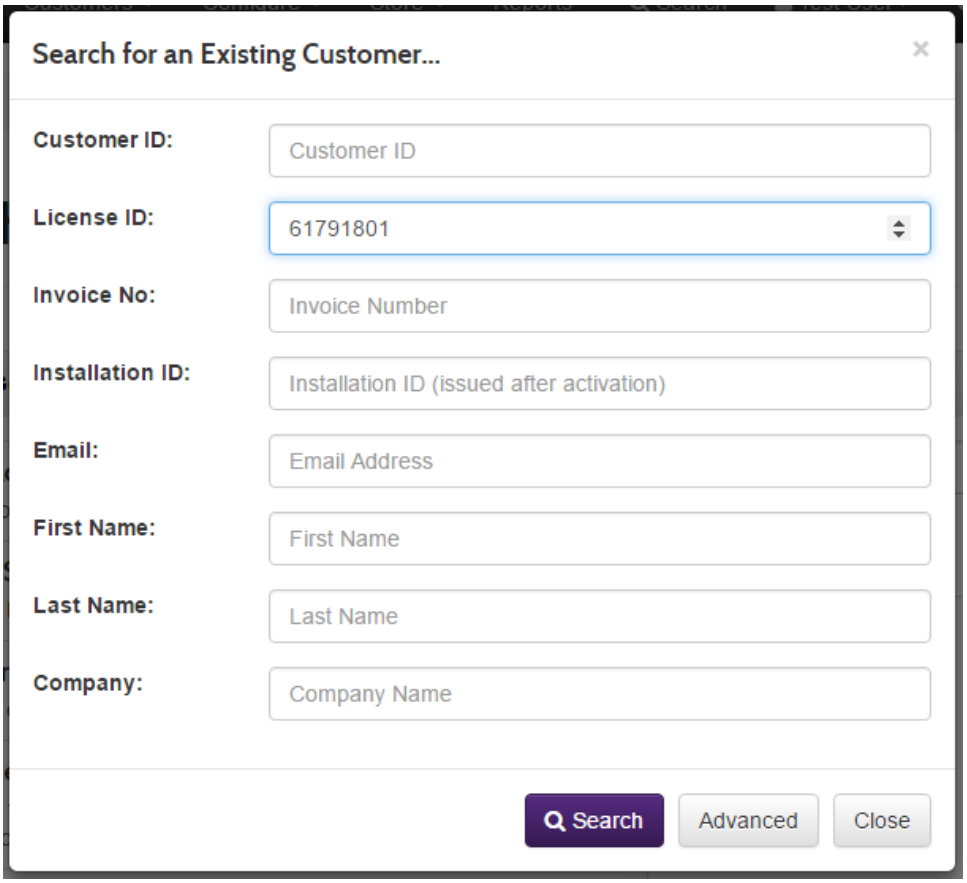
Once the License Status dialog appears, click deactivate license. A confirmation will appear if the license was successfully deactivated. The License Status will now show "License not found – activation is required".



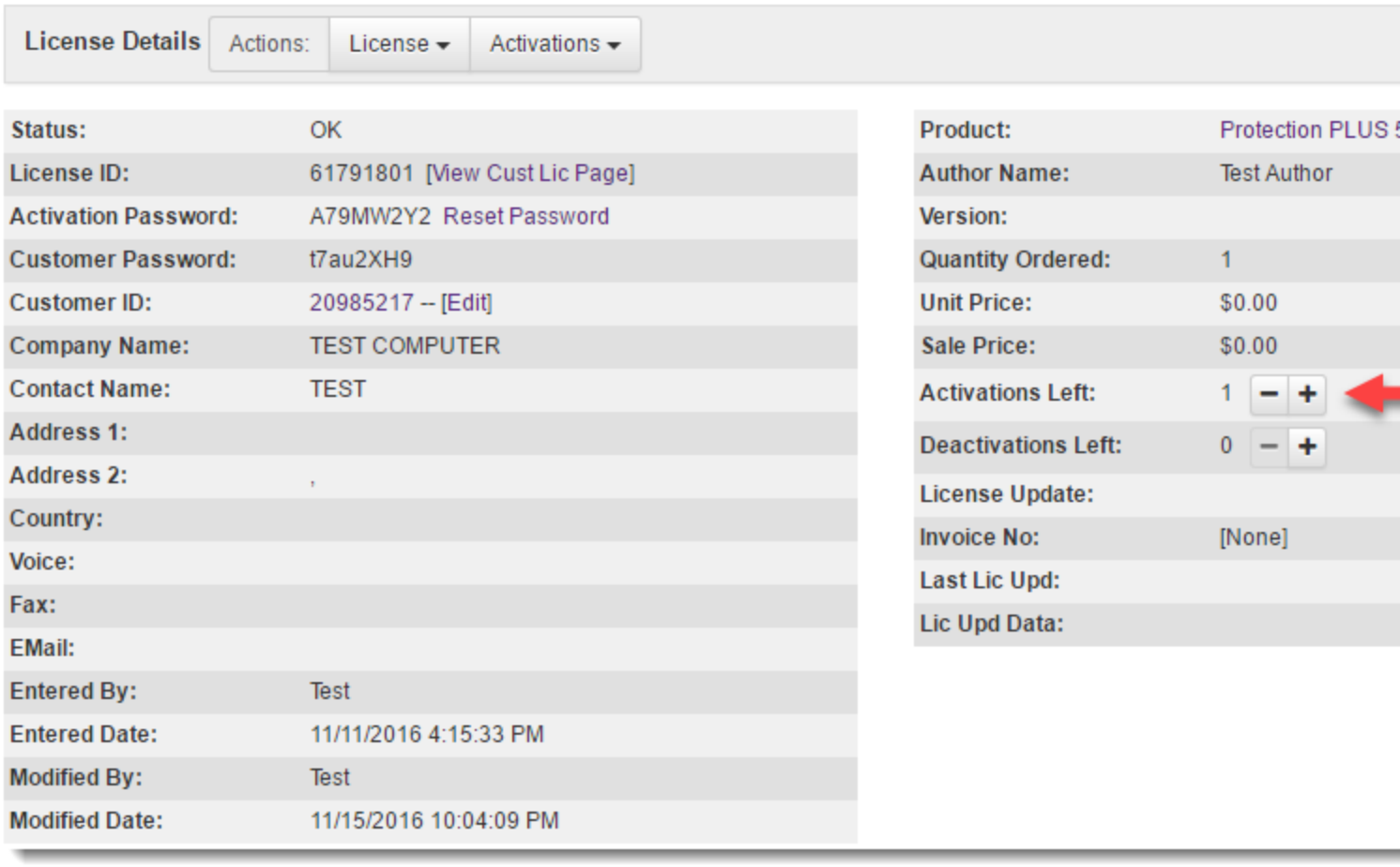
Click Activate Manually to proceed with testing a manual activation. A manual activation allows for activations from another computer or by e-mail. Follow the instructions in Step # 2 to generate a new license. Alternatively, we can also check the number of activations remaining for the License ID generated when we tested automatic activations online using Instant SOLO Server.



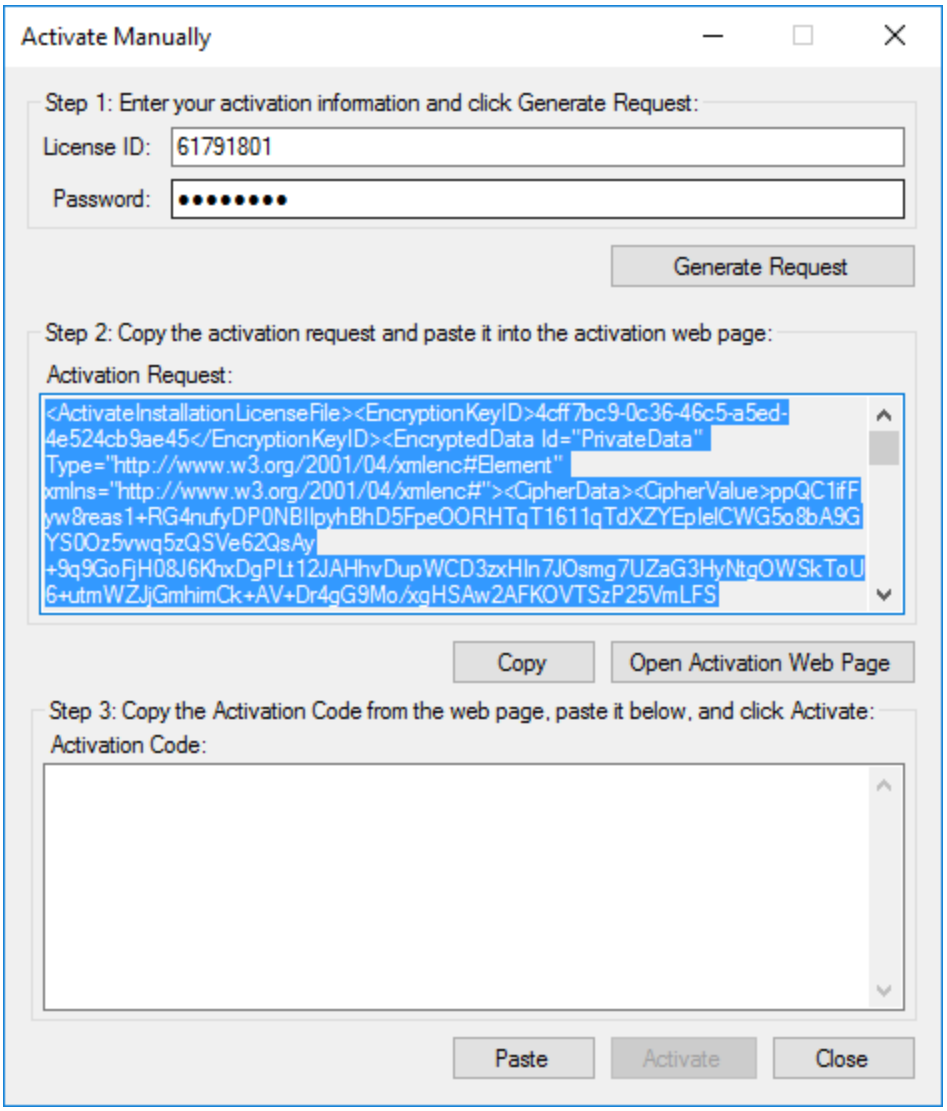
Return to Instant SOLO Server using the Test Author account. Perform a quick search for the License ID previously used to activate online (If you no longer have this number, proceed to create a new License ID as shown in Step # 2).



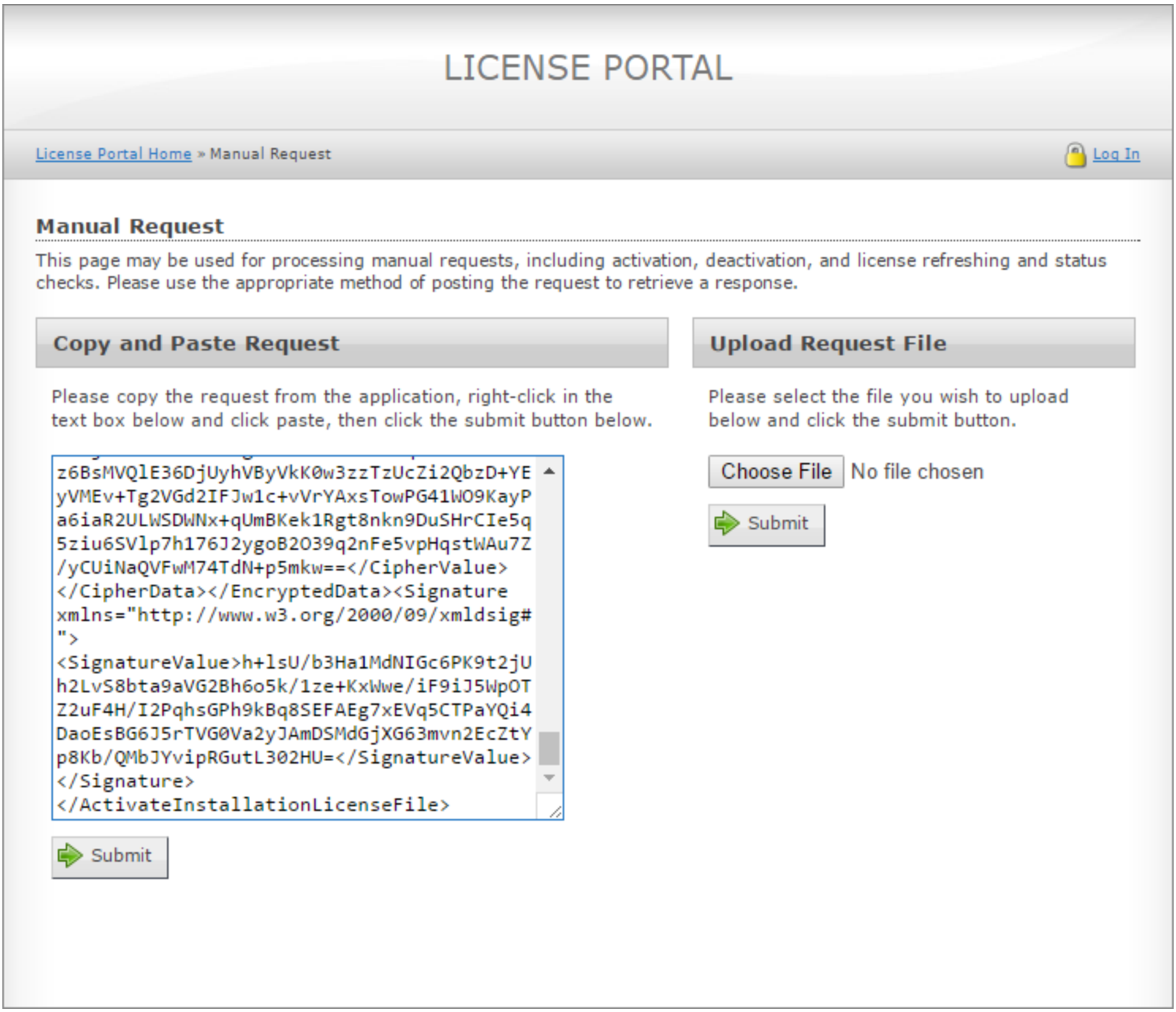
When we clicked on deactivate license, Instant SOLO Server automatically incremented the amount of activations left by 1. If you need to manually increase the number of activations left, you can increment this number by clicking the + button to the right of the Activations Left field.



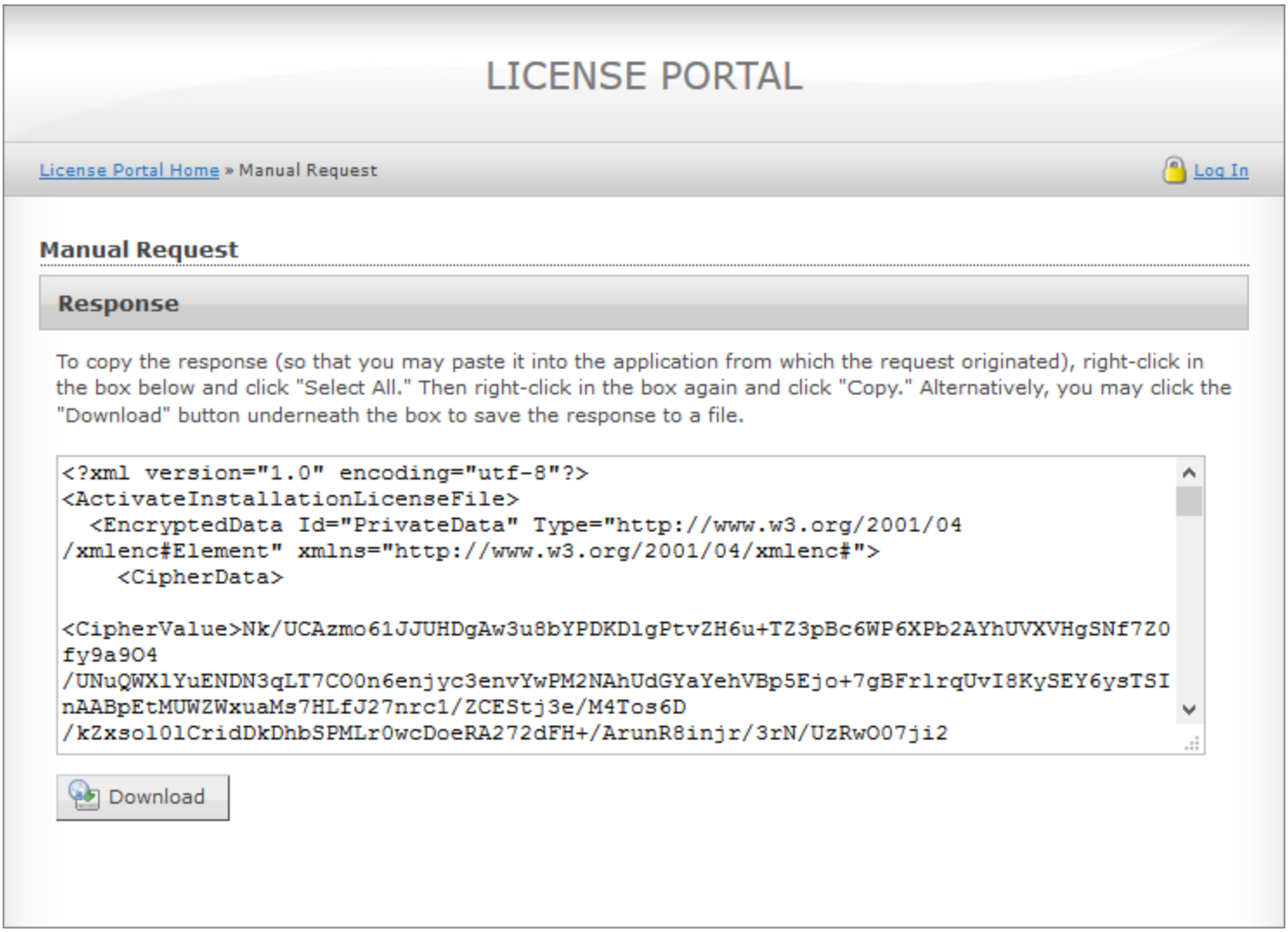
Return to the Activate Manually for our Simple Text Editor Sample. Enter the License ID and Password in their respective fields and click Generate Request.



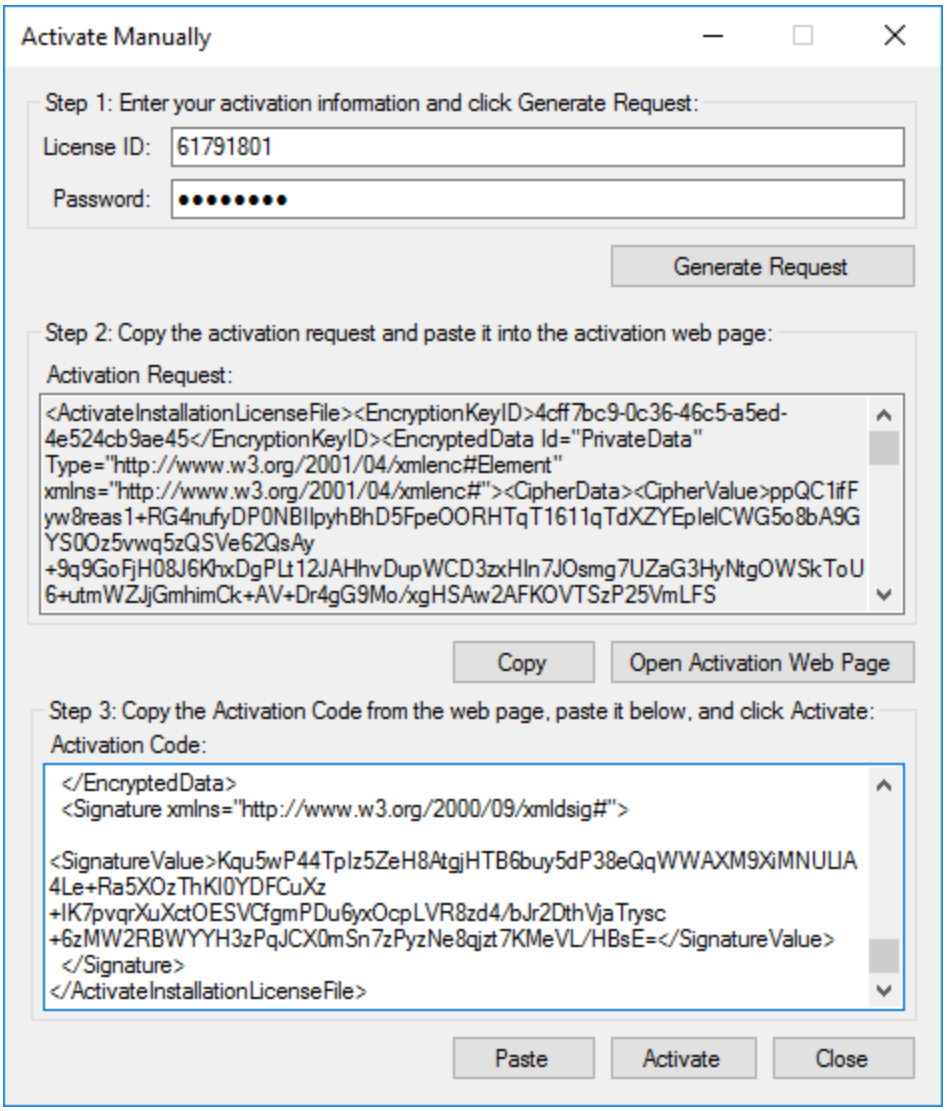
This dialog will generate an encrypted activation request to be processed by the activation server. Click Copy to copy this data to the clipboard. Next, click Open Activation Web Page. This web page is accessible from another computer with access to the Internet in the event that an offline workstation needed to be activated. Once you arrive to the License Portal page, paste the encrypted activation request data, and click Submit.



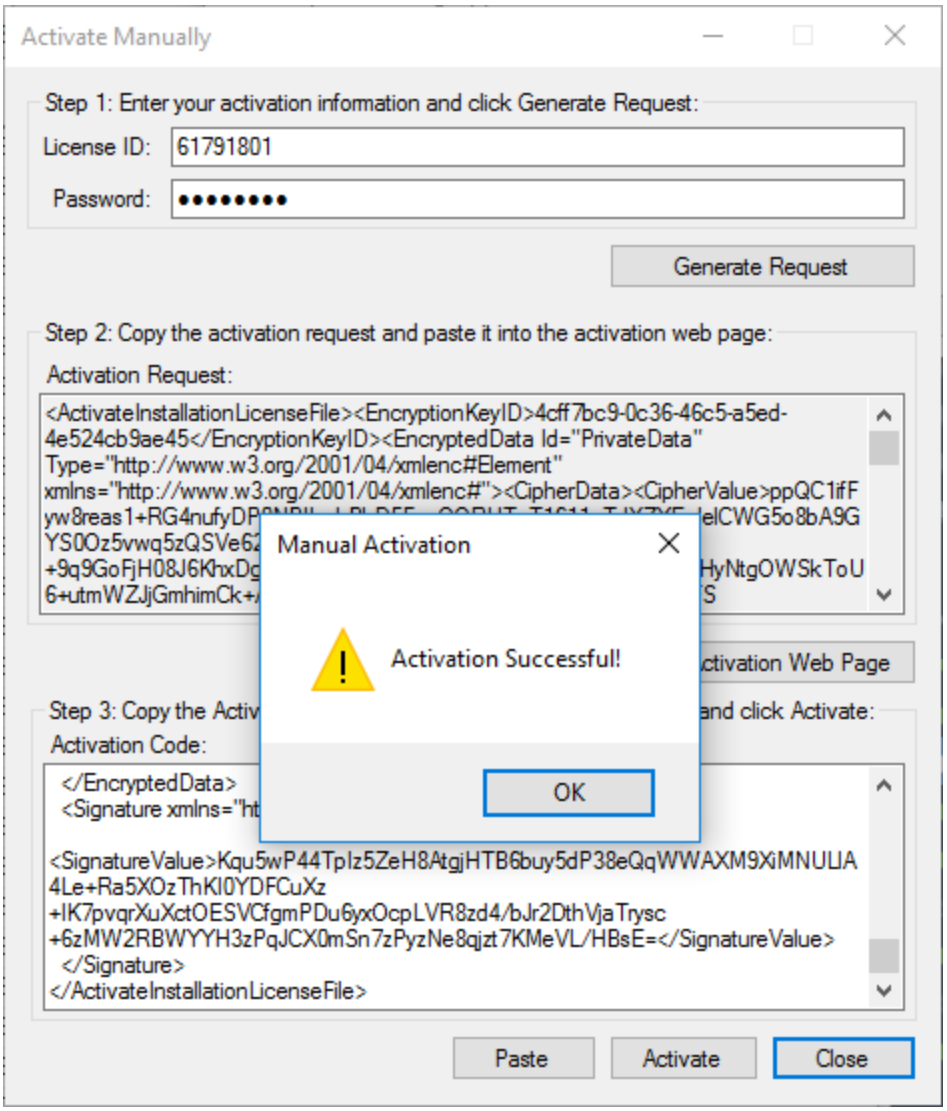
The License Portal will now generate an encrypted response. Copy this data to later paste in the Activation Code field of the activation window.



Return to the activation window and paste the encrypted response into the Activation Code field and click Activate.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.

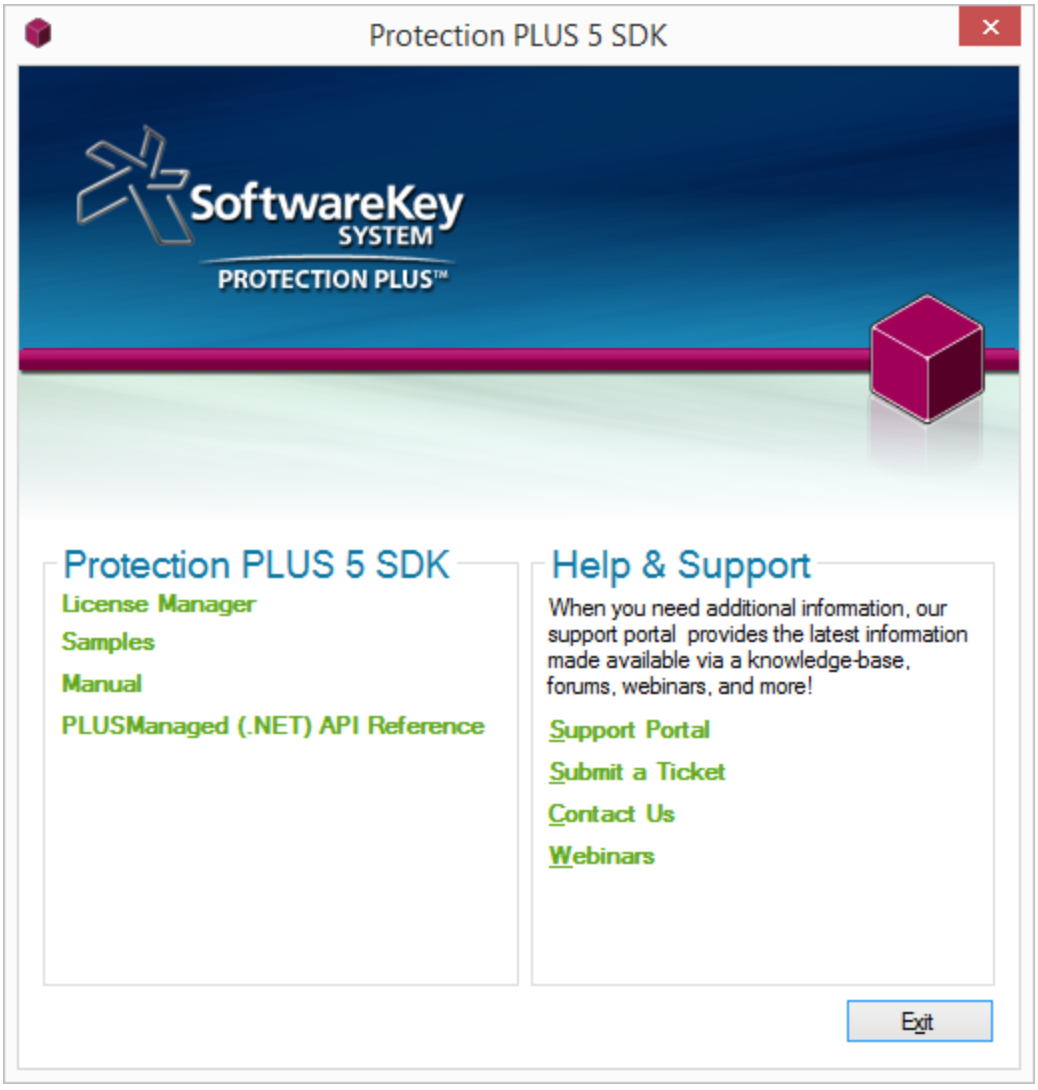


The License Status will now show as Fully Licensed along with any features previously enabled or disabled.

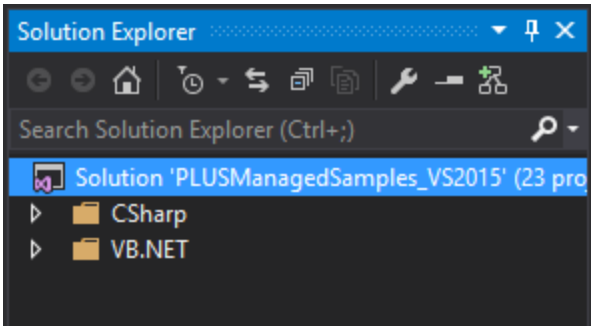
PLUSManaged Cloud-Controlled Network Floating Licensing with SOLO Server

Step 1 – Opening the Sample Project

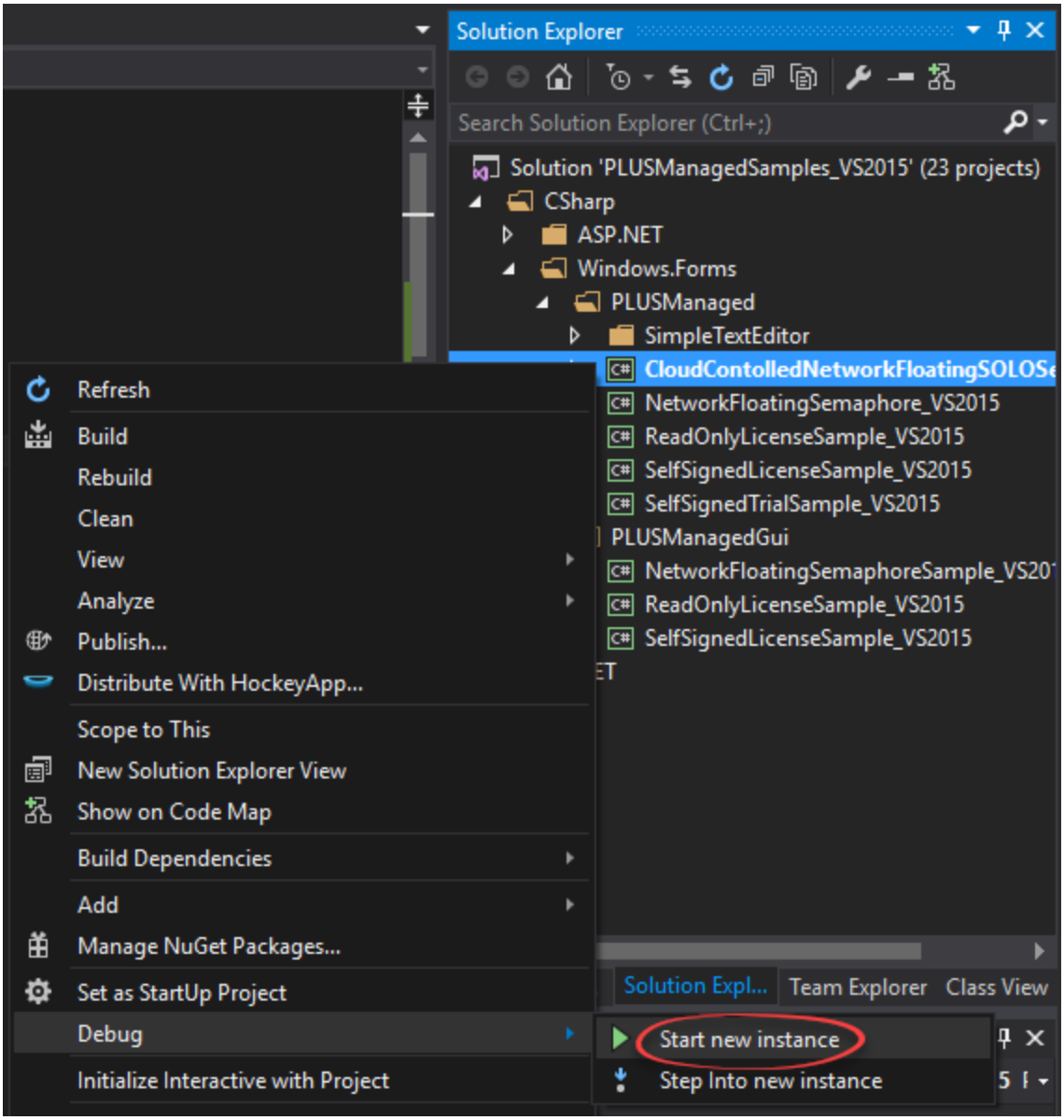
Open the PLUSManaged Samples *.sln file corresponding to your version of Visual Studio. The Samples directory can be opened by running Protection PLUS 5 SDK from your start menu/screen, and clicking on the Protection PLUS 5 SDK Samples link.



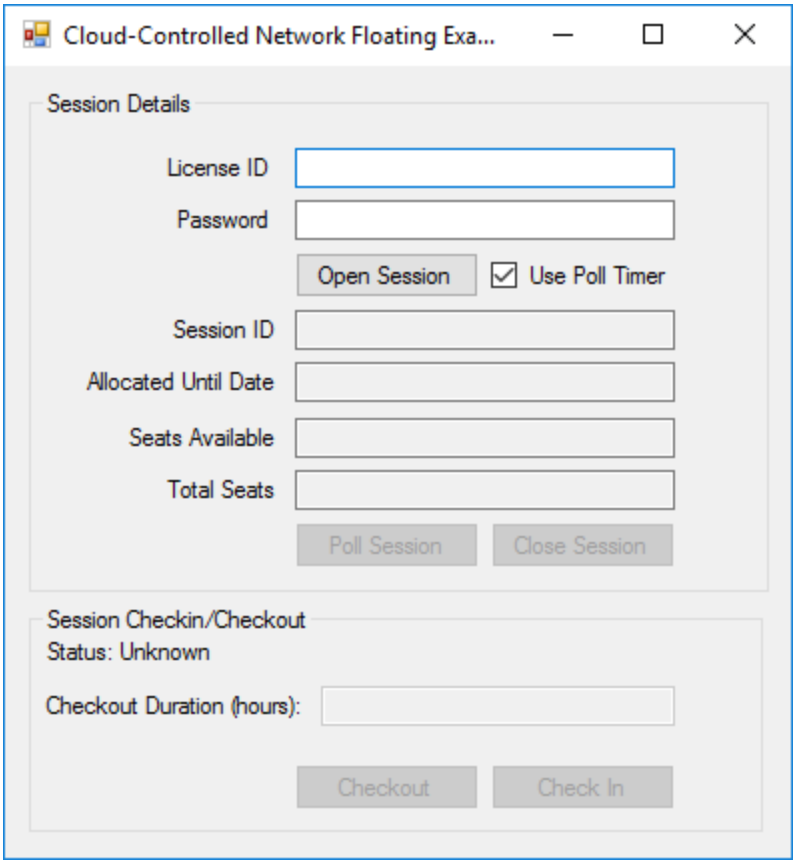
Once the project has loaded in Visual Studio, expand the programming language you wish to proceed with (C# or VB.NET).



Expand the Windows.Forms folder and PLUSManaged folder. Locate the CloudContolledNetworkFloatingSOLOServer sample. Right click this sample and select Debug > Start New Instance



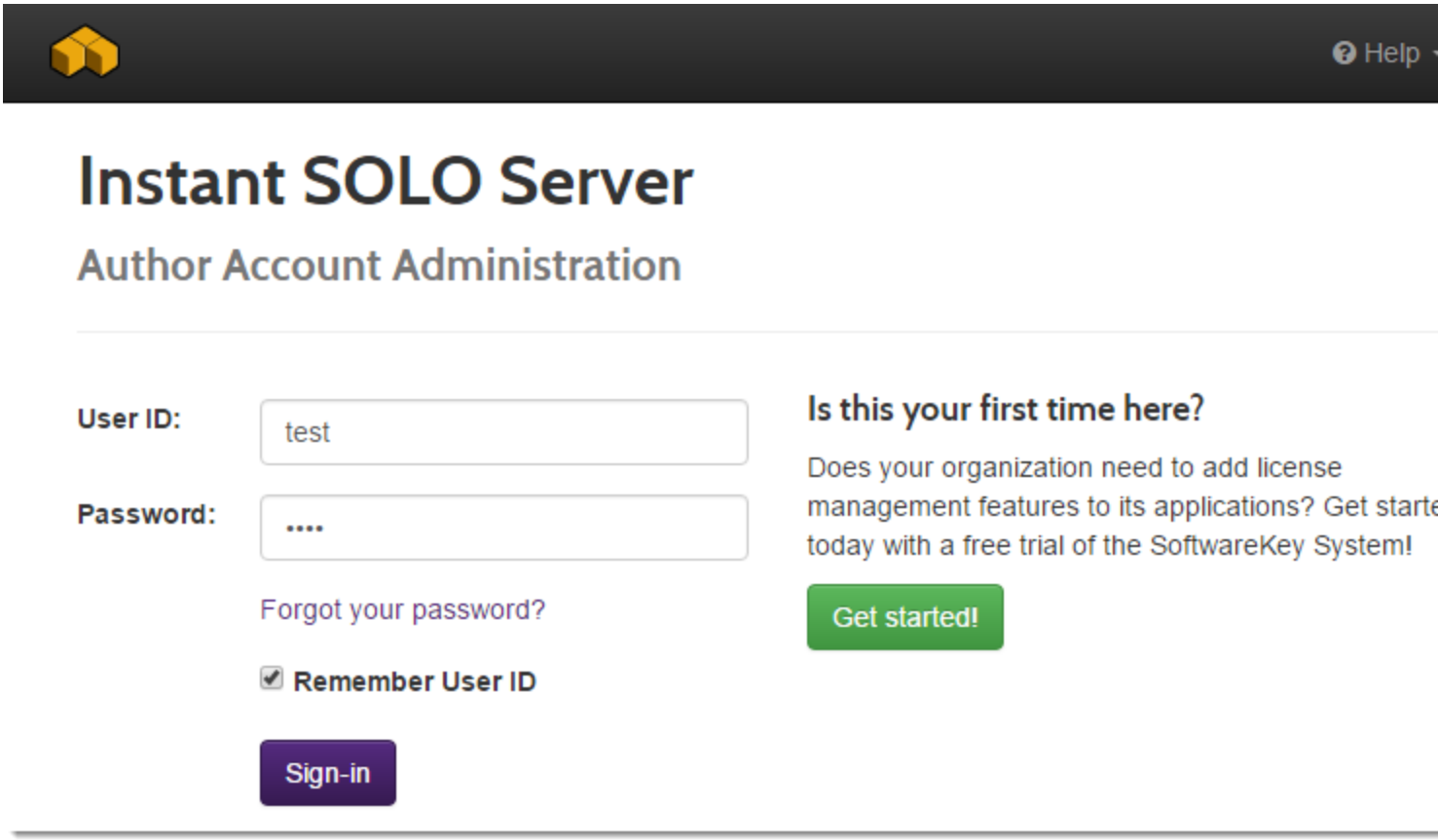
The sample application will execute.



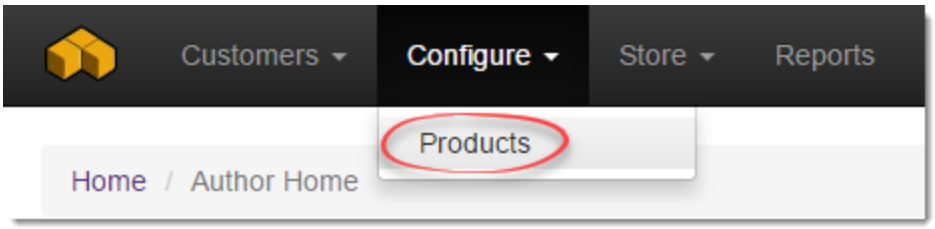
Step 2 – Opening a session with Instant SOLO Server

For ease of use, all of the Protection PLUS 5 SDK sample applications use a generic Test Author account with its own *Encryption Key ID* . We will use this Test Author account on Instant SOLO Server to generate the License ID and activation password necessary to activate the Sample. To log into Instant SOLO Server, visit **https://secure.softwarekey.com/solo/authors/Default.aspx**. Use the Test Author credentials given below:

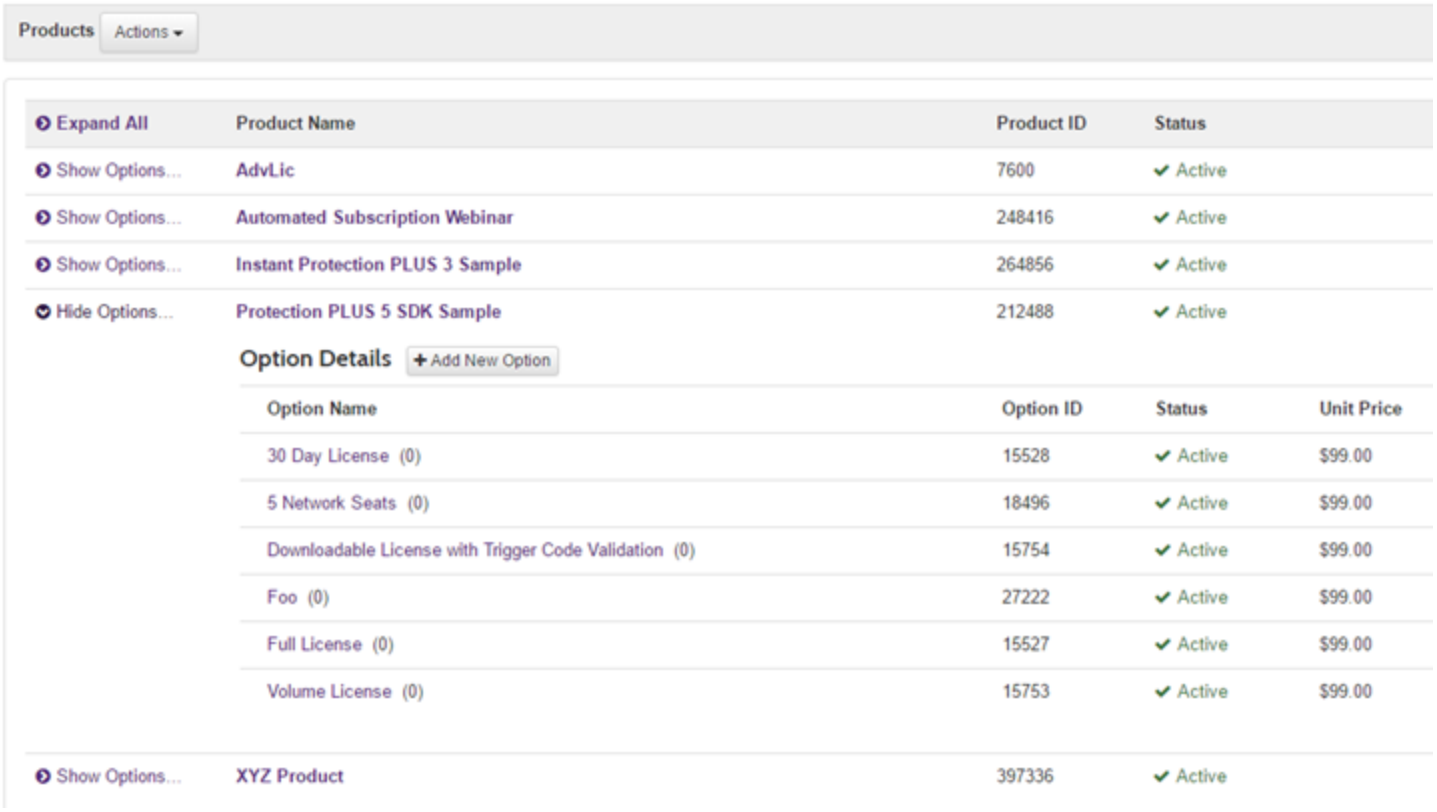
- User ID: test
- Password: test



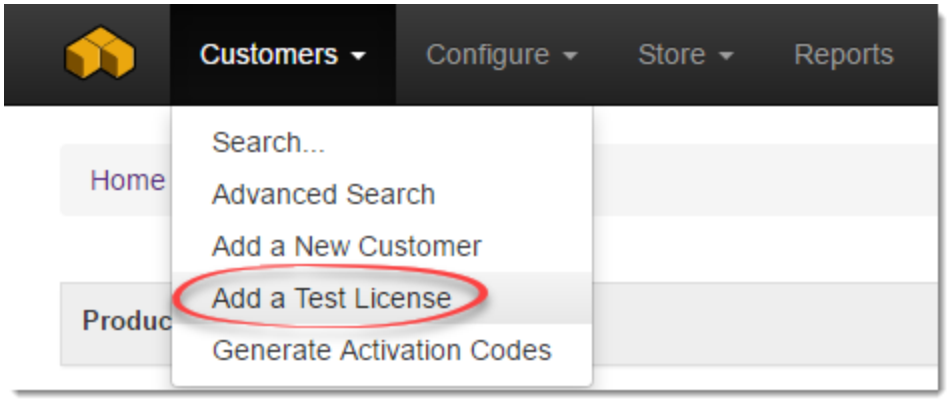
Once logged in, go to the menu *Configure / Products*.



The Product List page will show all of the products available to the Test Author. Expand Show Options beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for activating the sample application. For this tutorial, we will generate a license for the 5 Network Seats.

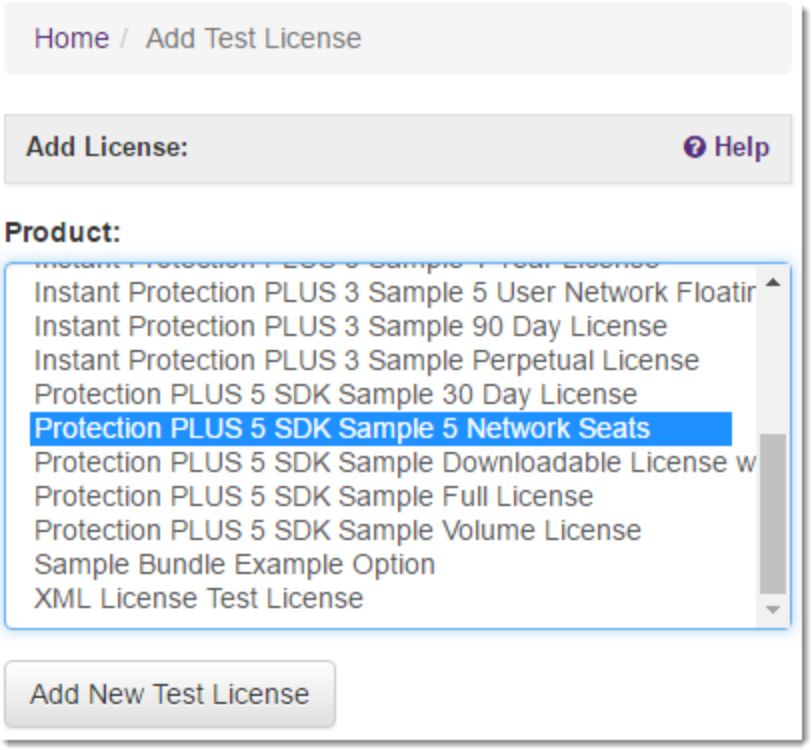


To add a test license for this product option, mouse over Customers in the navigation bar, and click "Add Test License".



>

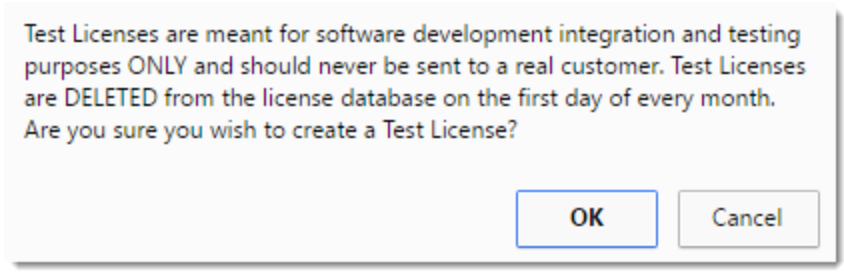
Select the Protection PLUS 5 SDK Sample 5 Network Seats. Type a customer password of your choice in the password field below, and click Add New Test License.



Important:

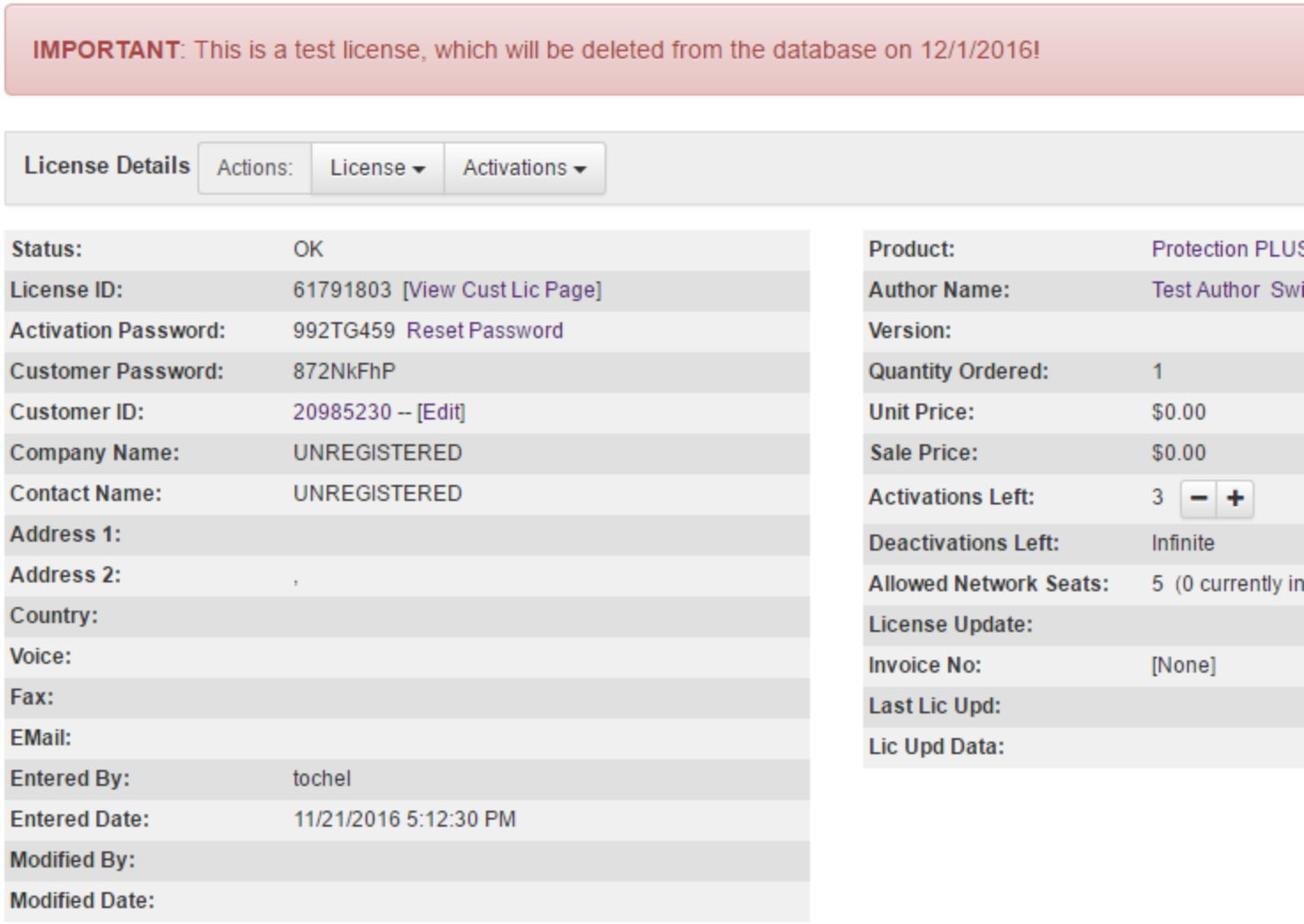
Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.

Press OK to continue when prompted.

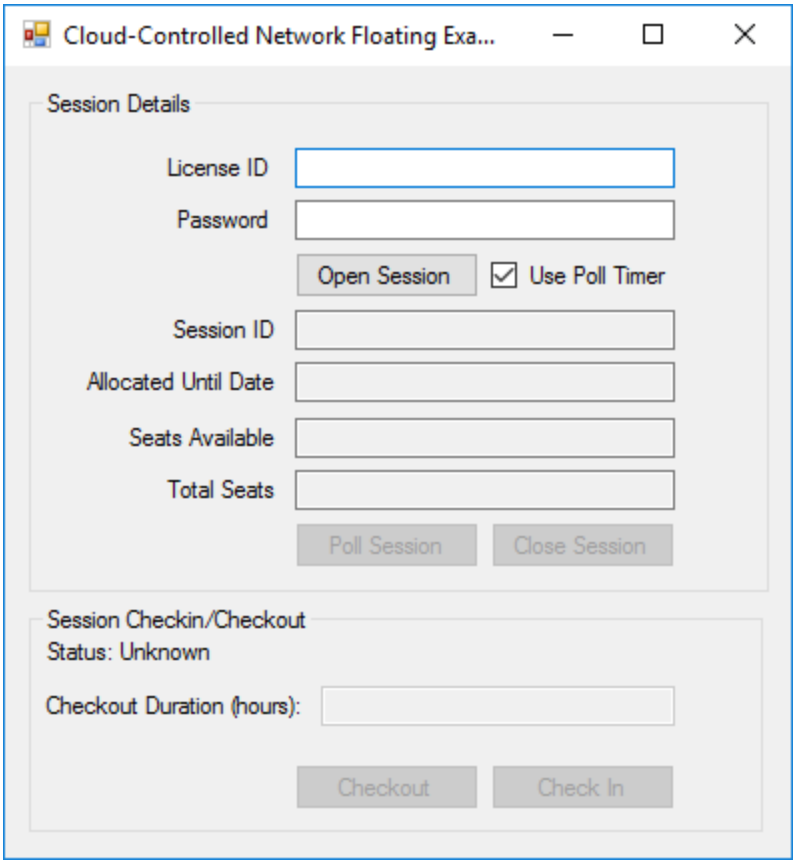


>

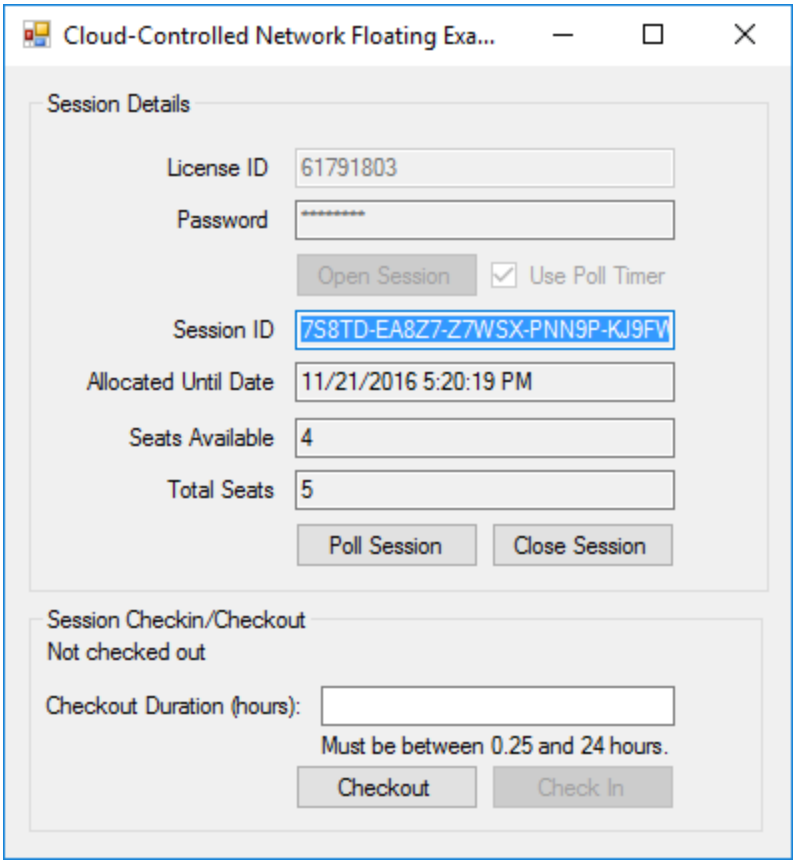
The Test License has now been created. We now have a License ID and an Activation Password to continue testing the sample application.



Return to the Cloud-Controlled Network Floating Licensing Sample application. Enter the License ID and Activation Password in their respective fields. Click Open Session to communicate with Instant SOLO Server for license validation and to open a session.



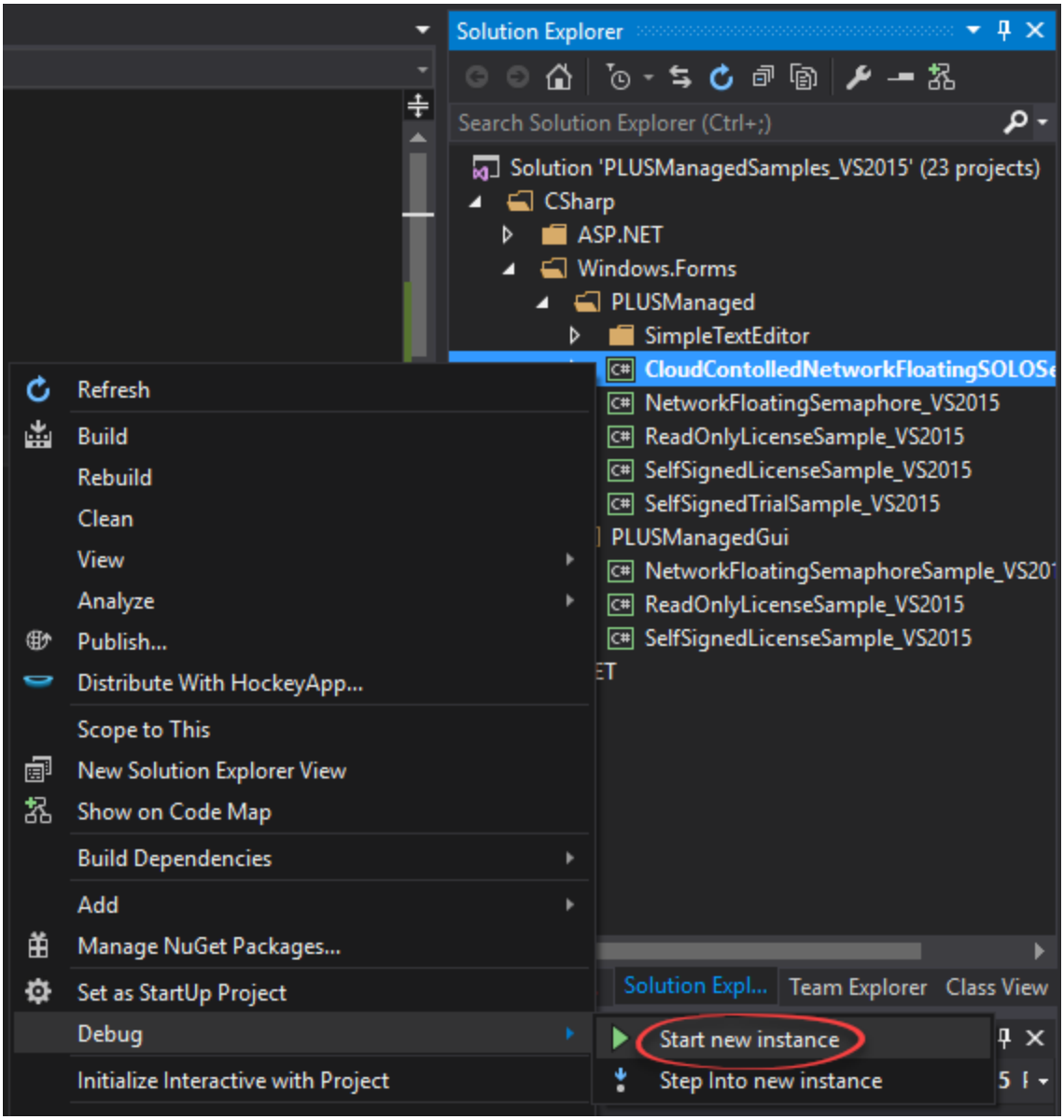
Upon successful activation, the Session ID, Allocated Until Date, Seats Available, and Total Seats edit boxes will be propagated with values received from Instant SOLO Server. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



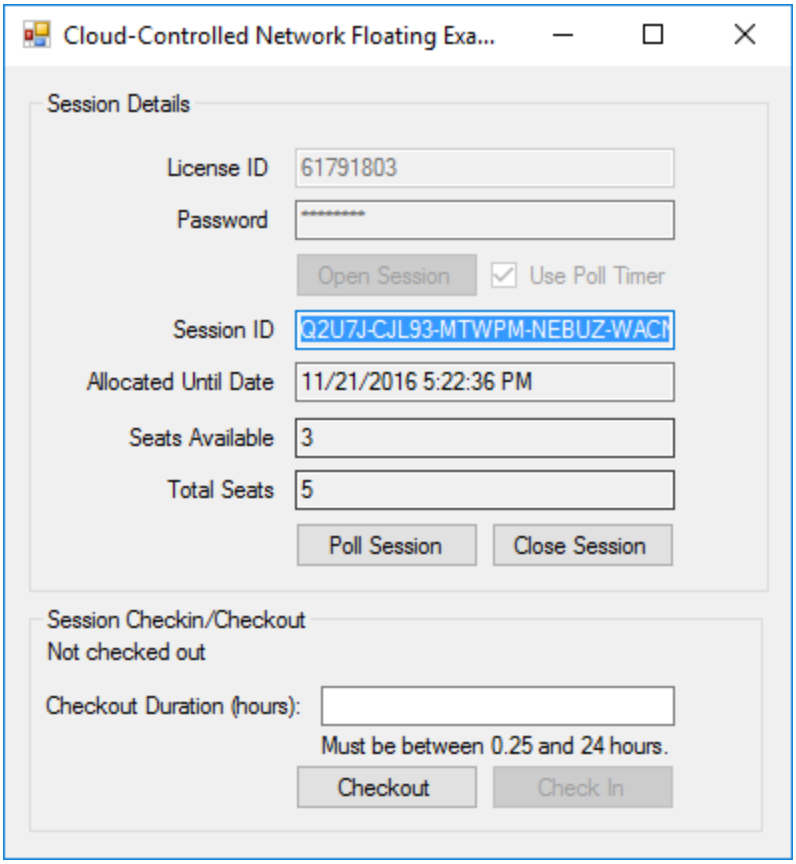
Note the number of Seats Available is 4 out of 5 since we have just used one of the seats with this session.

Step 3 – Opening a Second Session

Open another instance of the sample application, but do not close the instance that is currently open. In Visual Studio right click this sample and select Debug > Start New Instance



To open a session for the second instance of the sample application, enter the same License ID and Activation Password used in the previous instance into their respective fields in this second instance. Click Open Session to communicate with Instant SOLO Server for license validation and to open a session.



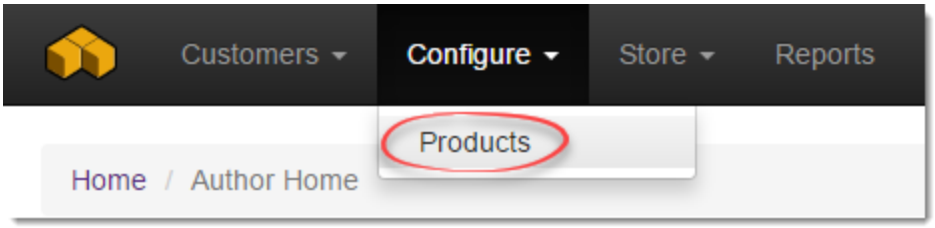
Note there are now 3 seats available of 5 total seats.

Step 4: Finding the Settings in SOLO Server

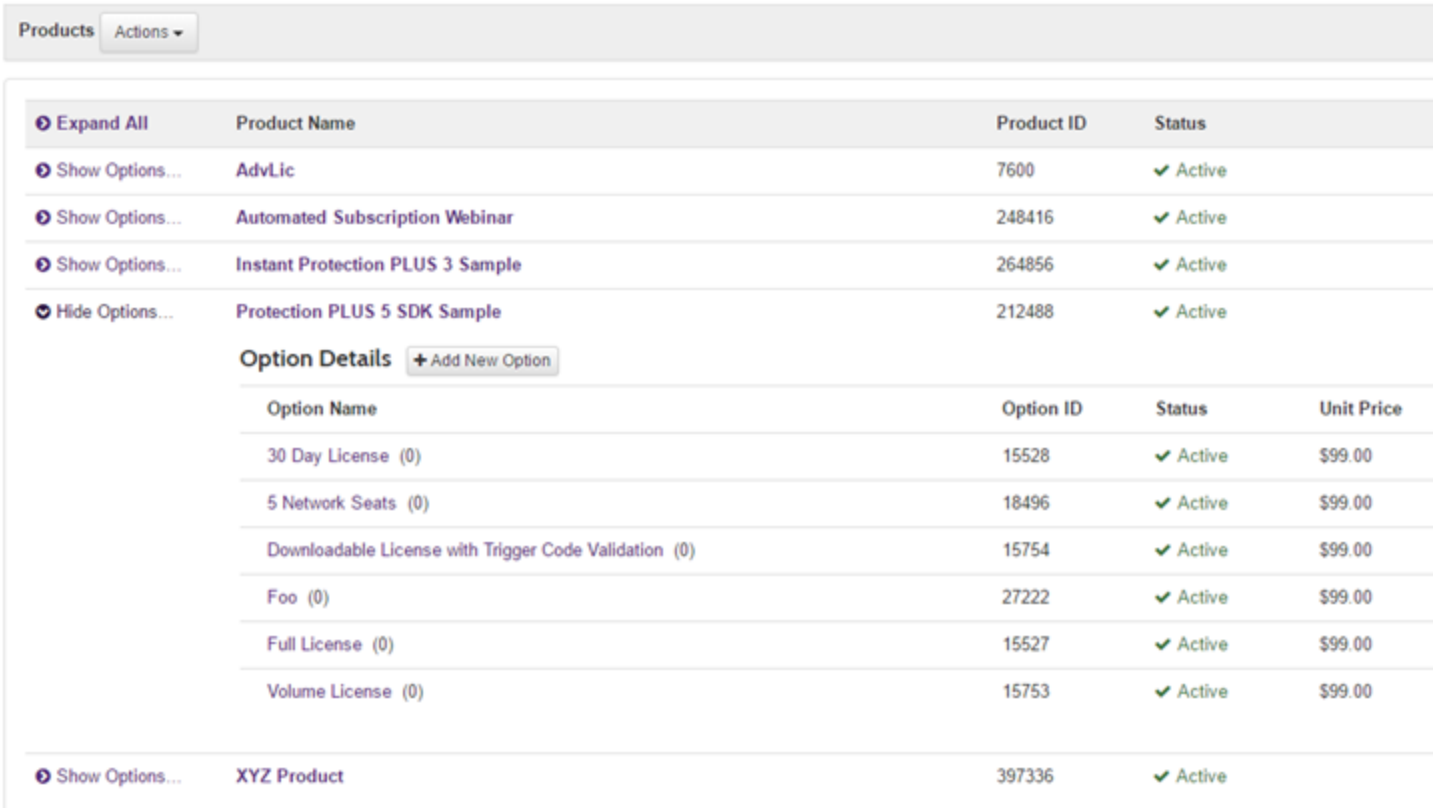
Important:

These settings cannot be accessed with the Test Author account, you will be able to access this **ONLY** when your account is fully activated and enabled for Cloud-Controlled Network Floating Licensing.

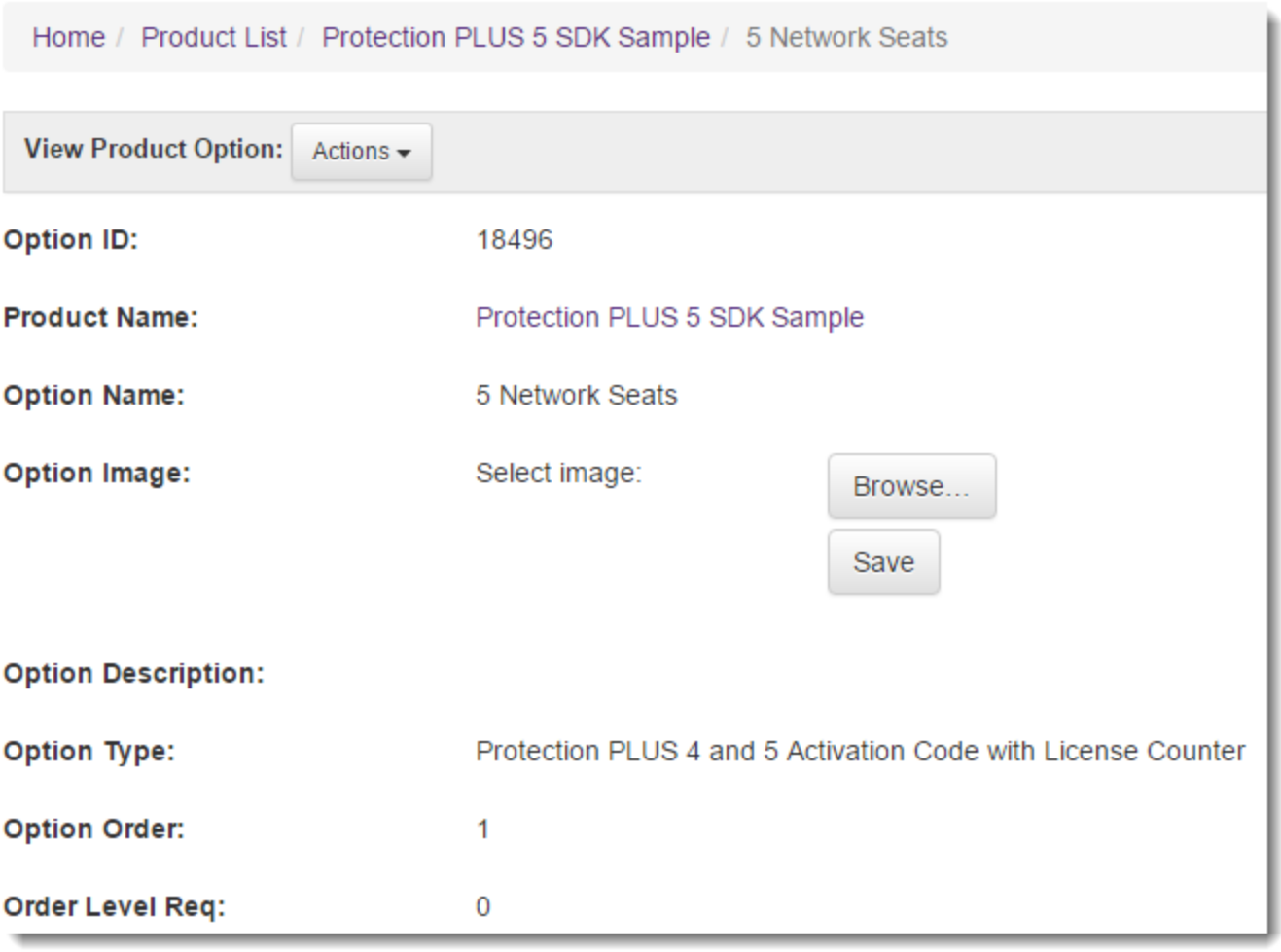
Returning to SOLO Server again, go to the menu *Configure / Products*.



The Product List page will show all of the products available to the Test Author. Expand Show Options beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for the sample applications.



Click on the link for the 5 Network Seats option in Instant SOLO Serverwhich will take you to the details page for this option.



Click the Actions dropdown and choose Edit next to "View Product Options:".

Product Activation and Licensing Details:

Activations per U/M: 3

☑ **Issue License**

☑ **Issue Installation ID**

☑ **Allow Deactivations**

Deactivations per U/M: -1

(-1=Unlimited)

☐ **Allow Reactivations on the Same Computer**

Please review the documentation

IP Activation Mode: No restrictions

IP Mask: Please Select

Additional Activation Days: 0

Minimum Activation Version:

Minimum Activation URL:

Unlock Threshold: -1

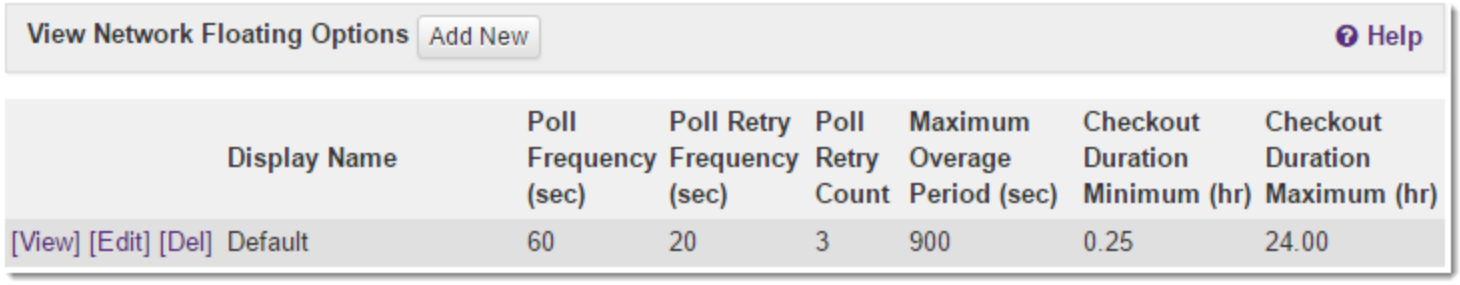
(advanced option)

Trigger Code #: 6

Trigger Code Seed: 400

RegKey 2 Seed: 123

Click on the Configure Network Floating Options link highlighted above.

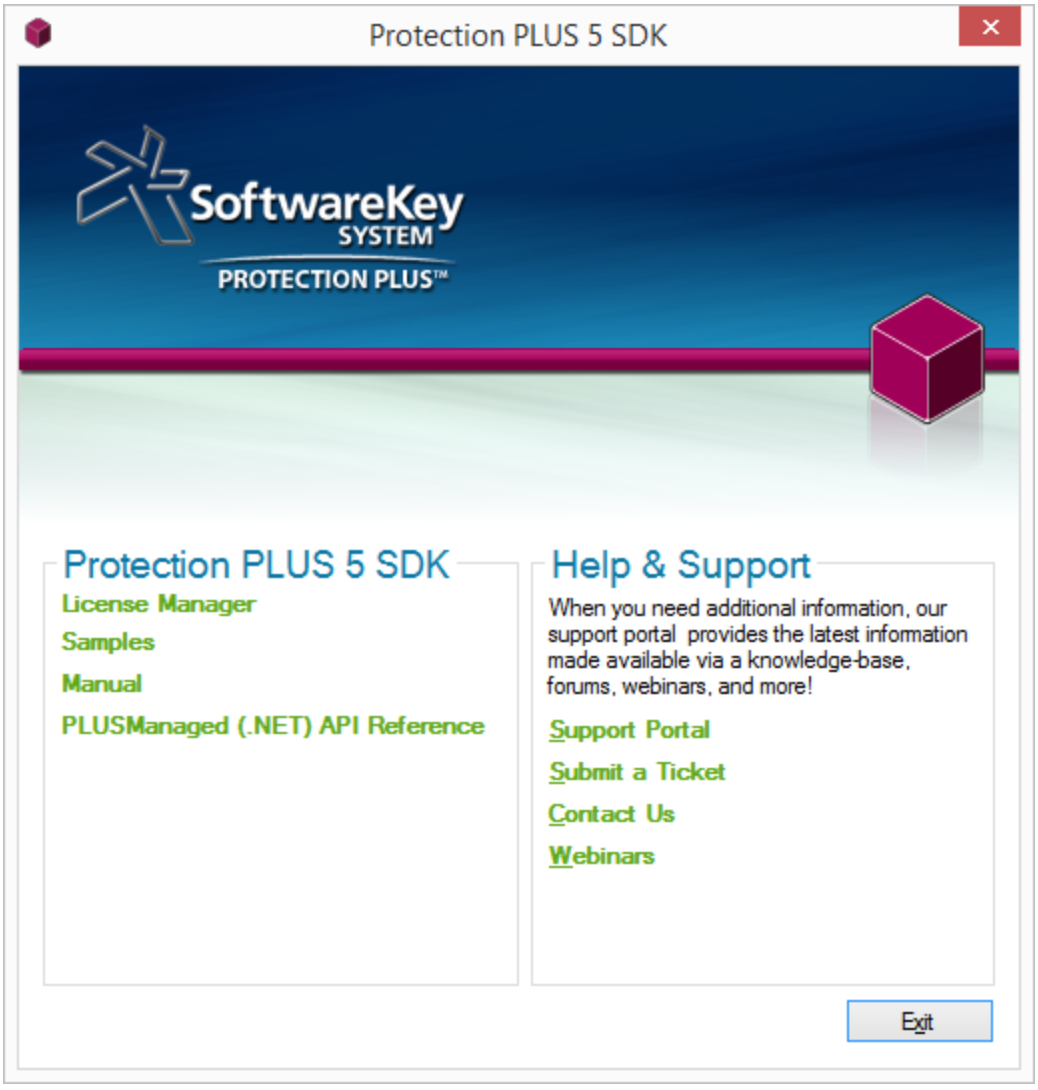


Here you can view and edit the settings SOLO Server is sending to the application when it opens a session.

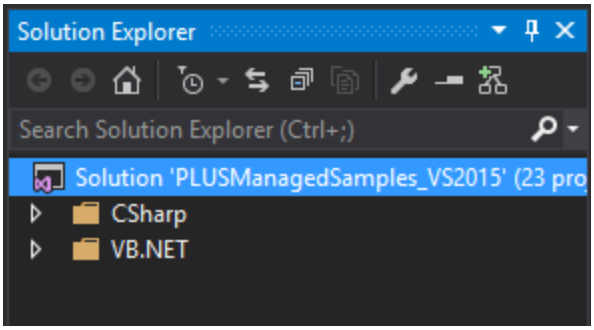
PLUSManaged ASP.NET Licensing Sample

Step 1 – Opening the Licensing Sample Project

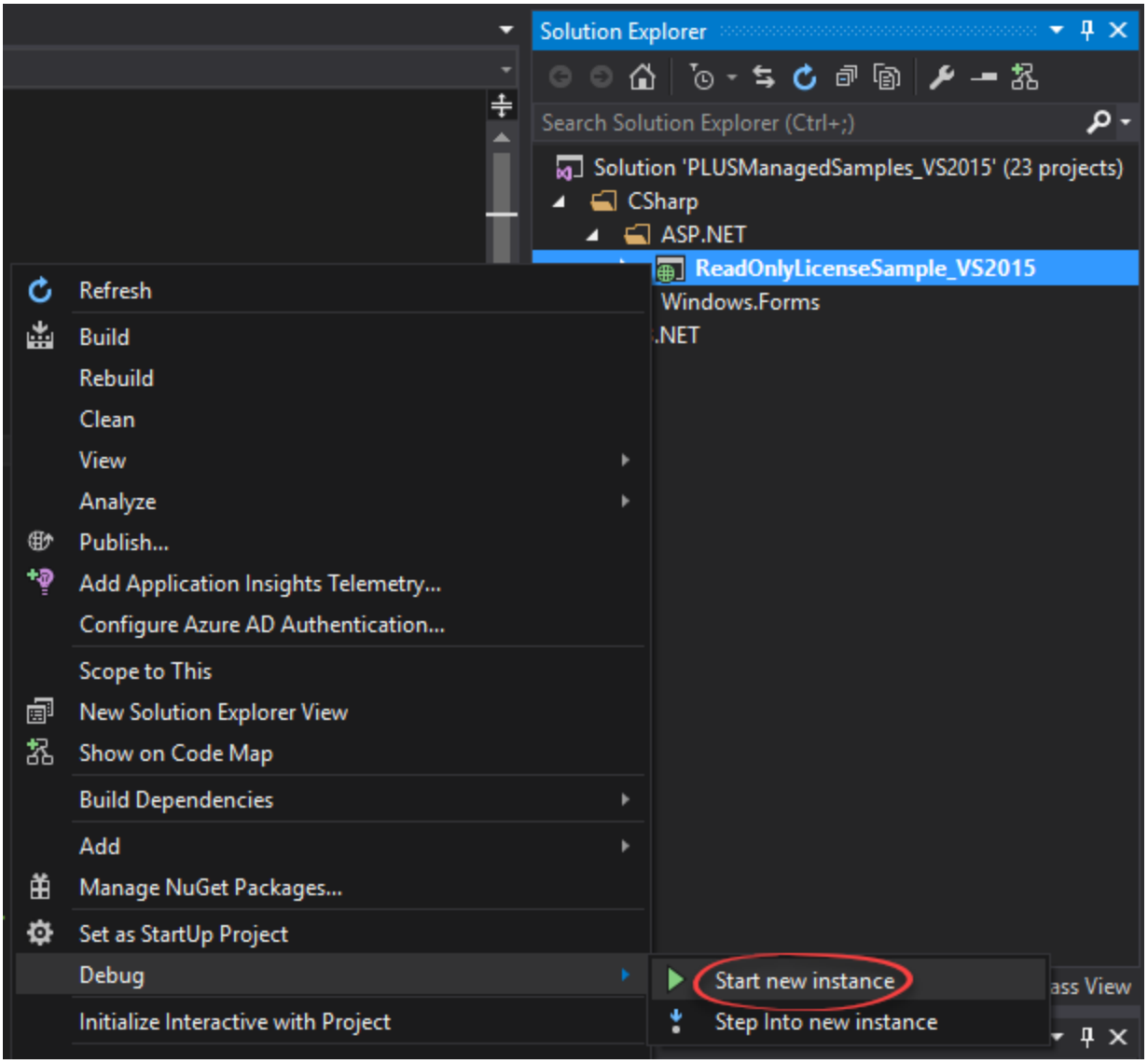
Open the PLUSManagedSamples *.sln file corresponding to your version of Visual Studio. The Samples directory can be opened by running Protection PLUS 5 SDKfrom your start menu/screen, and clicking on the Protection PLUS 5 SDK Samples link.



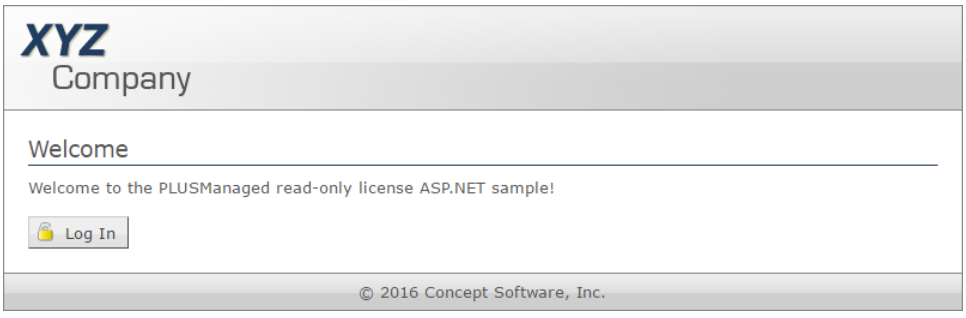
Once the project has loaded in Visual Studio, expand the programming language you wish to proceed with (C# or VB.NET).



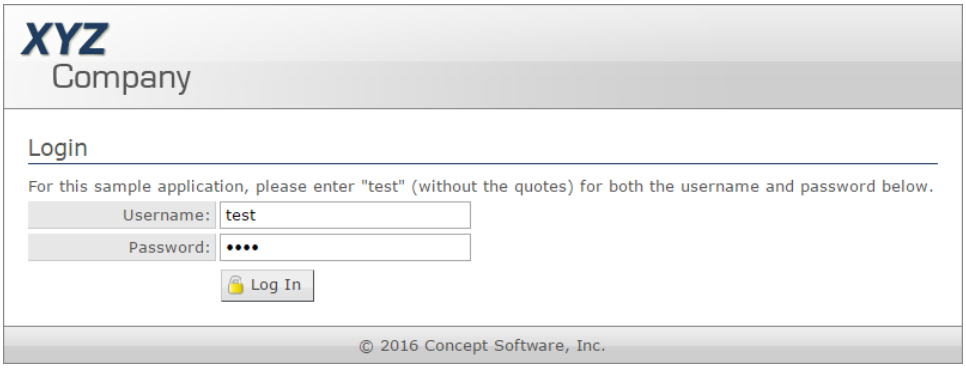
Expand the ASP.NET folder and locate the ReadOnlyLicense Sample. Right click this sample and select Debug > Start New Instance



Your browser will open to the sample's company login page. Click the Log In button.



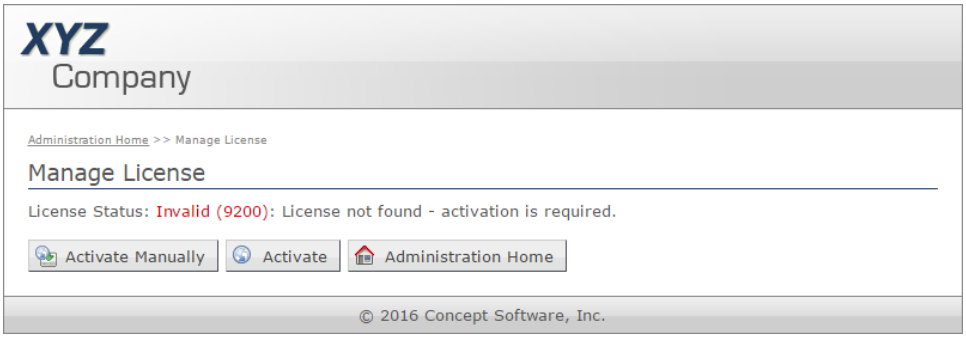
Enter the login credentials "test" (without the quotes) for both the Username and Password and click the Log In button.



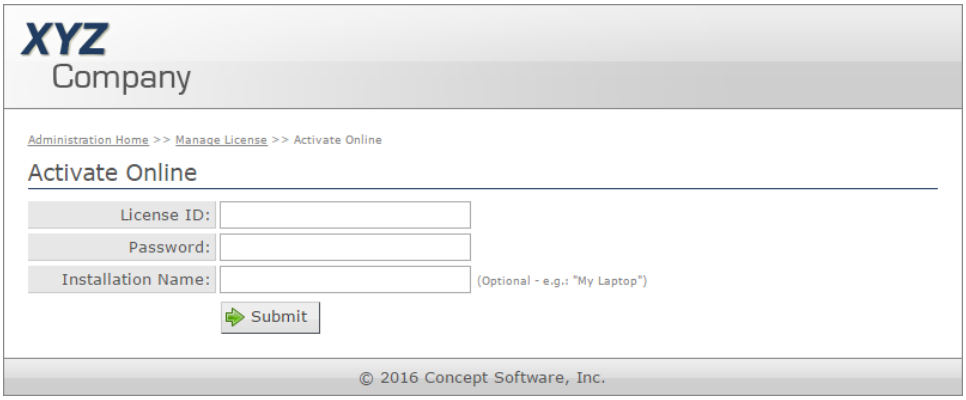
You will be taken to the Administration page and shown the license was not found, therefore an activation is required.

Step 2 – Activating automatically with Instant SOLO Server

To activate the ReadOnlyLicenseSample automatically using *Instant SOLO Server*, click the Manage License button to be taken to the Manage License page.

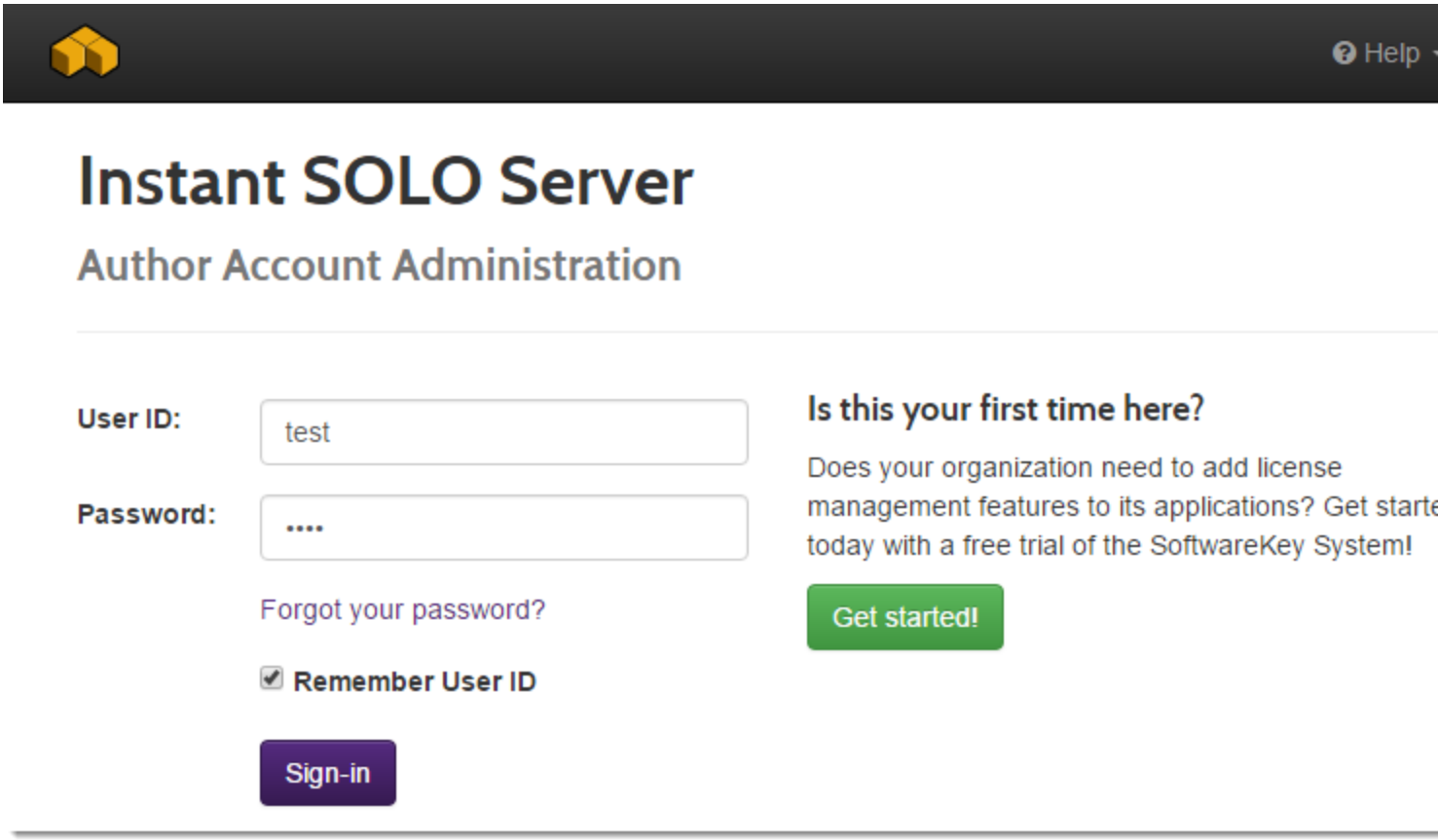


On the Manage License page click the Activate button to be taken to the Activate Online page where you are prompted for a License ID and Password (both are required). Specifying an installation name is optional, but helpful when performing multiple installations for different machines and/or users.

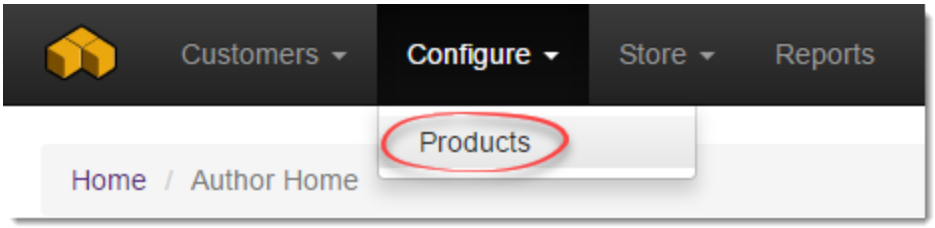


For ease of use, all of the Protection PLUS 5 SDK sample applications use a generic Test Author account with its own *Encryption Key ID* . We will use this Test Author account on Instant SOLO Server to generate the License ID and password necessary to activate the ASP.NET ReadOnlyLicensing Sample. To log into Instant SOLO Server, visit **https://secure.softwarekey.com/solo/authors/Default.aspx**. Use the Test Author credentials given below:

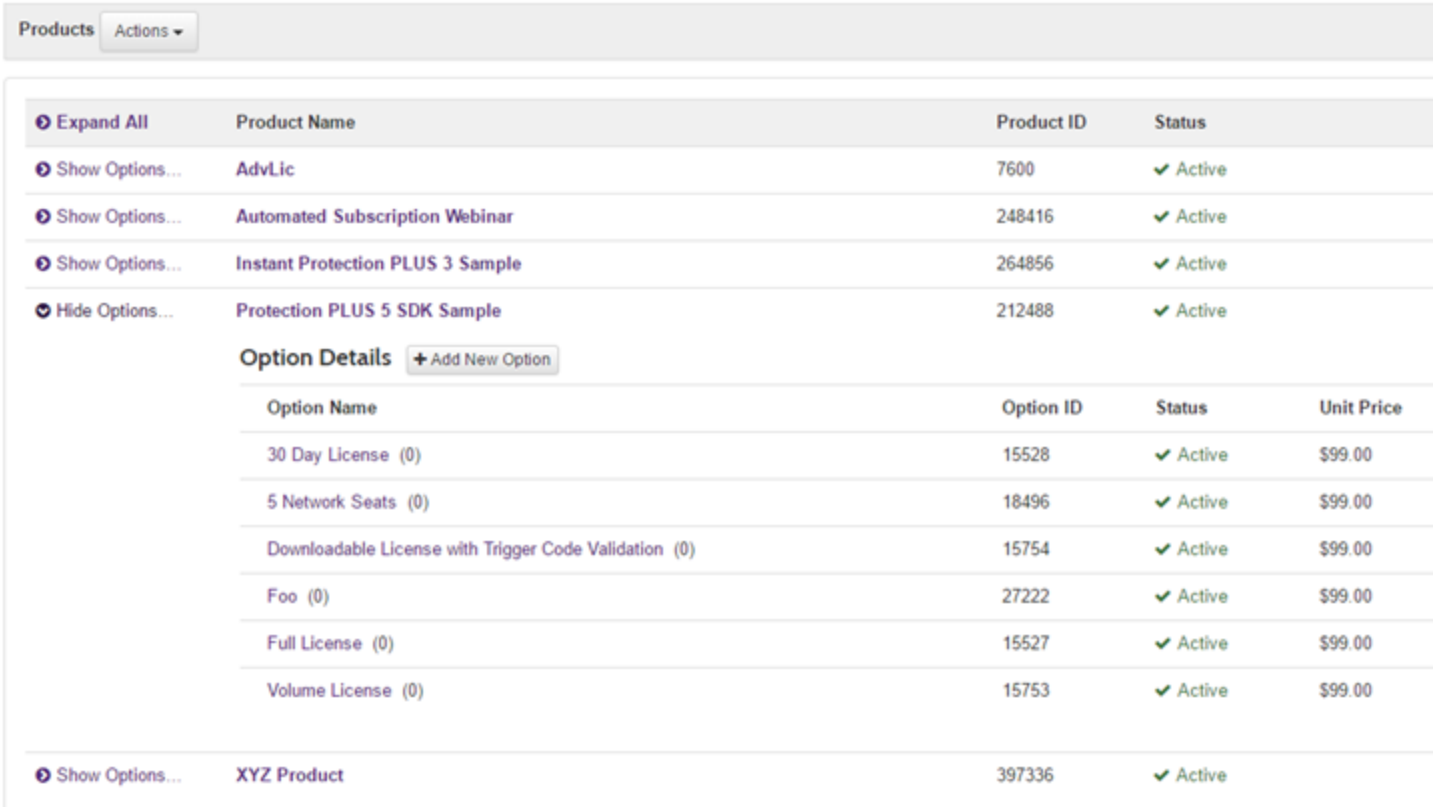
- User ID: test
- Password: test



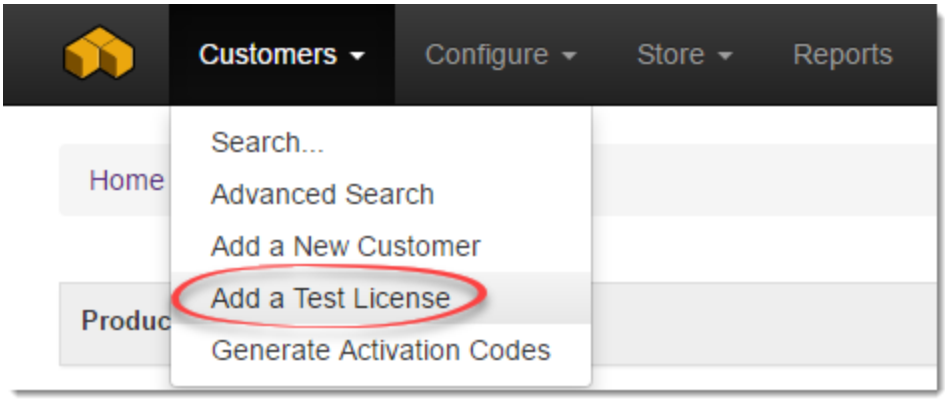
Once logged in, go to the menu *Configure / Products*.



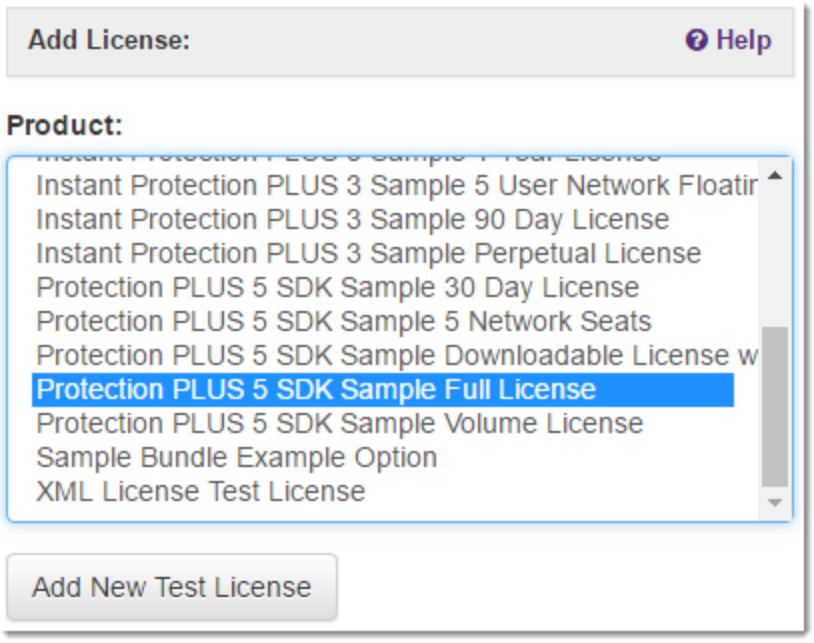
The Product List page will show all of the products available to the Test Author. Expand Show Options beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for activating the Licensing Sample application. For this tutorial, we will generate a license for the Full License.



To add a test license for this product option, mouse over Customers in the navigation bar, and click "Add Test License".



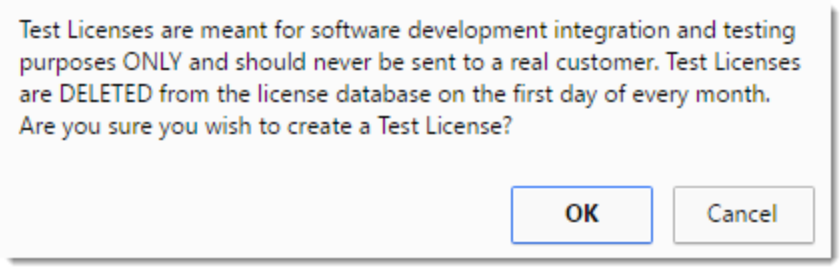
Select the Protection PLUS 5 SDK Sample Full License and click *Add New Test License*.



Important:

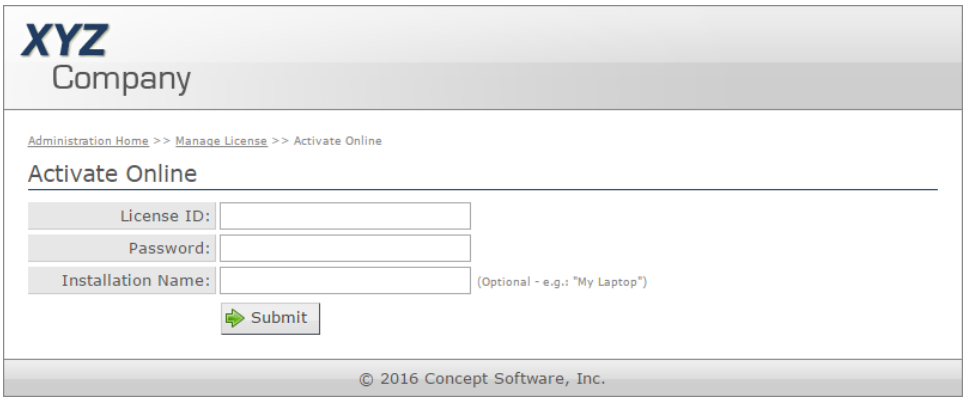
Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.

Press OK to continue when prompted.

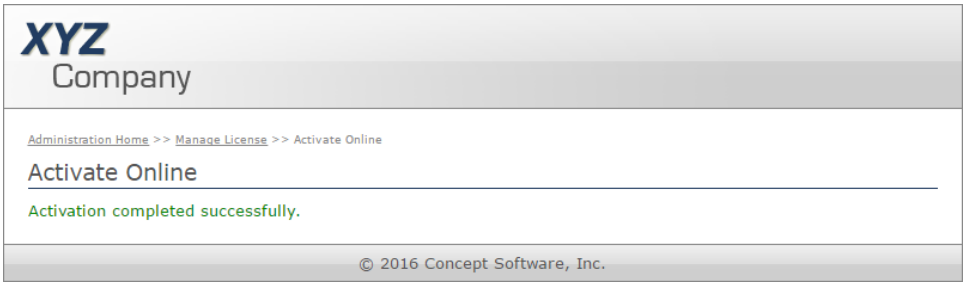


The Test License has been created. We now have a License ID and Password to continue testing the sample.

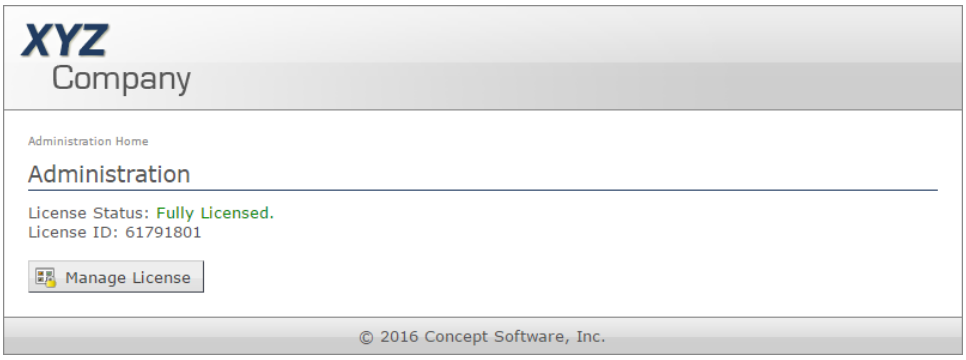
Return to the activation prompt for our ASP.NET ReadOnlyLicensing Sample. Enter the License ID and Password in their respective fields and, optionally, an Installation Name. Click Submit to communicate with Instant SOLO Server for license validation.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



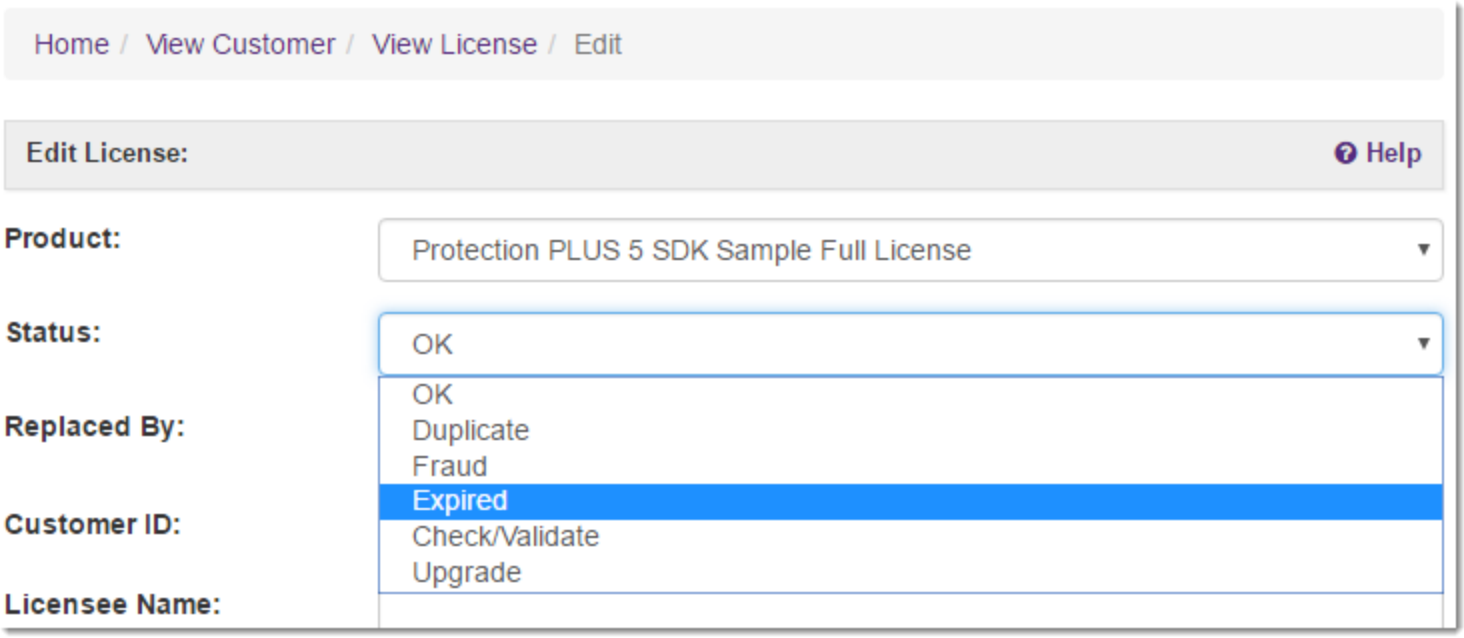
On the breadcrumb links, click on the Administration Home link and you will see the license status.



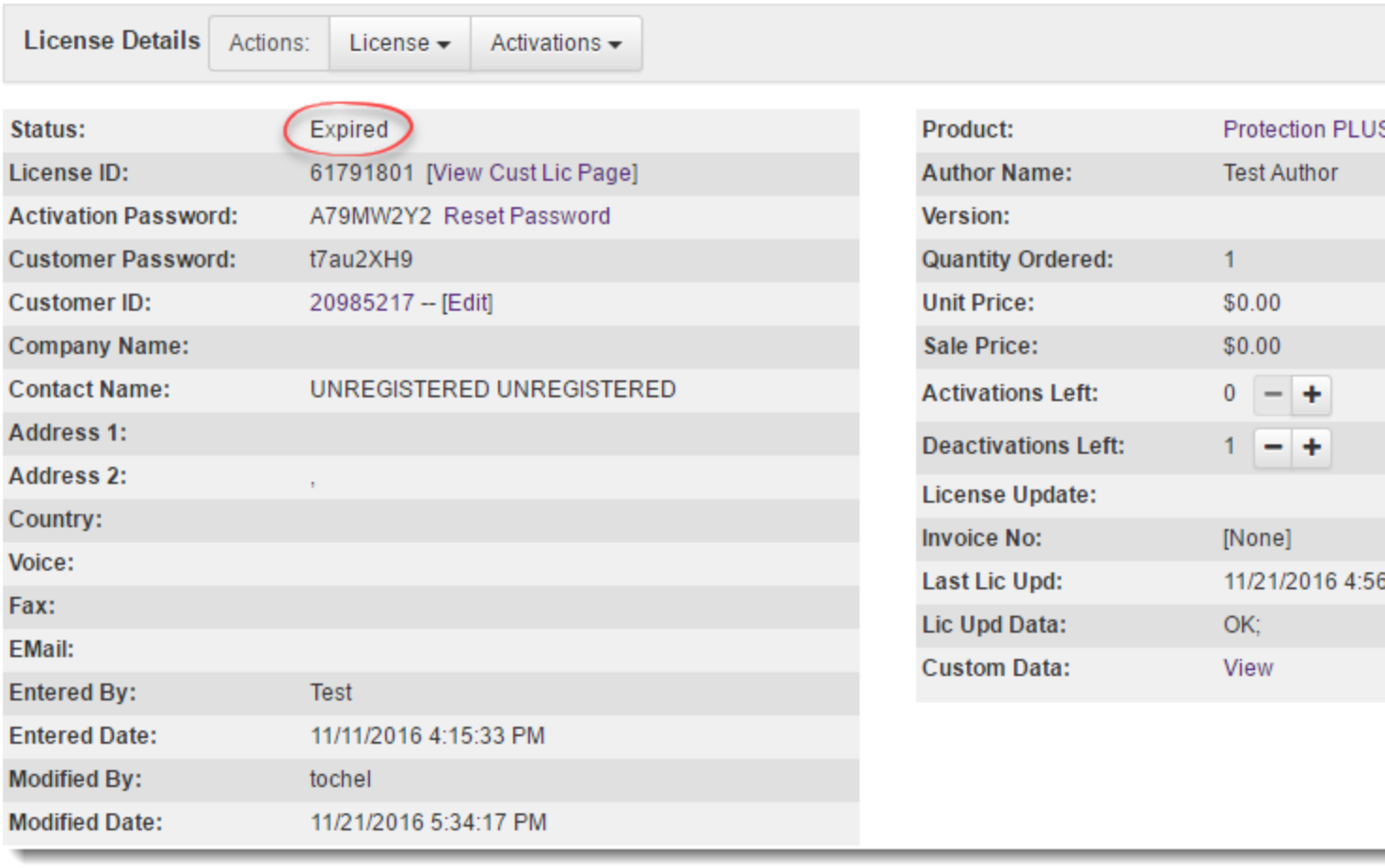
Step 3- Refresh a License

A license can be refreshed with updated information sent from Instant SOLO Server. We can modify the license status from the License Details page on Instant SOLO Server in order to remotely deactivate the license. On the License Details page click the Actions dropdown and choose *Edit*.

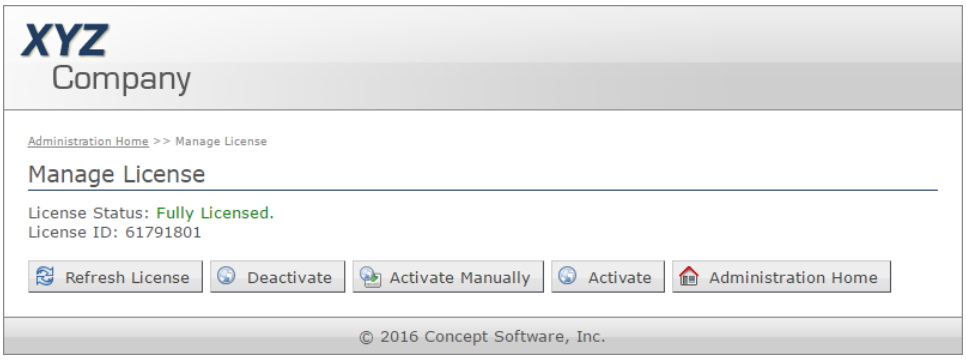
Change the Status value by clicking on the dropdown and select Expired (as this is a test license there are some status values not shown, and you can click Help in the top right of the Instant SOLO Server page to see a full list). Scroll to the bottom and click Submit.



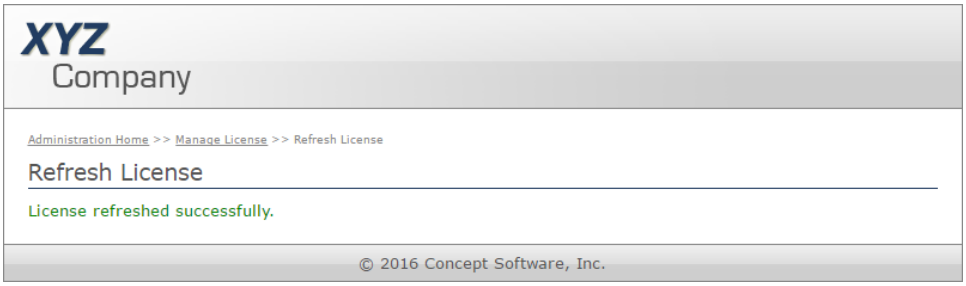
The page returns to the License Details page. This page now displays the Status as Expired.



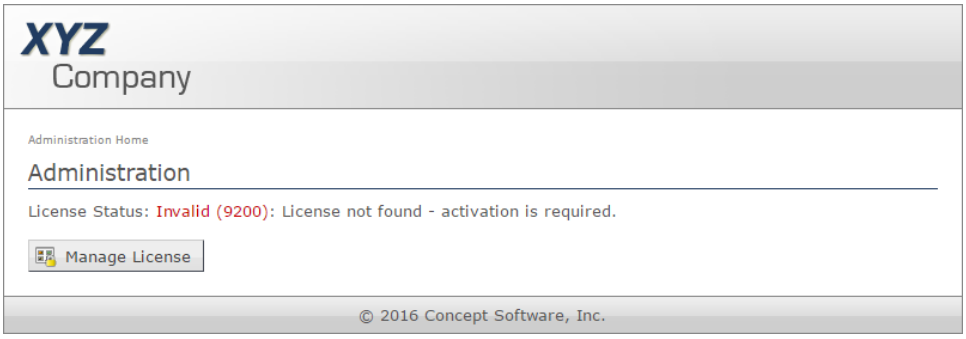
Return to the ASP.NET ReadOnlyLicense Sample and click Manage License to be taken to the Manage License page.



Click Refresh License. If the licensed successfully refreshed a message will be displayed.

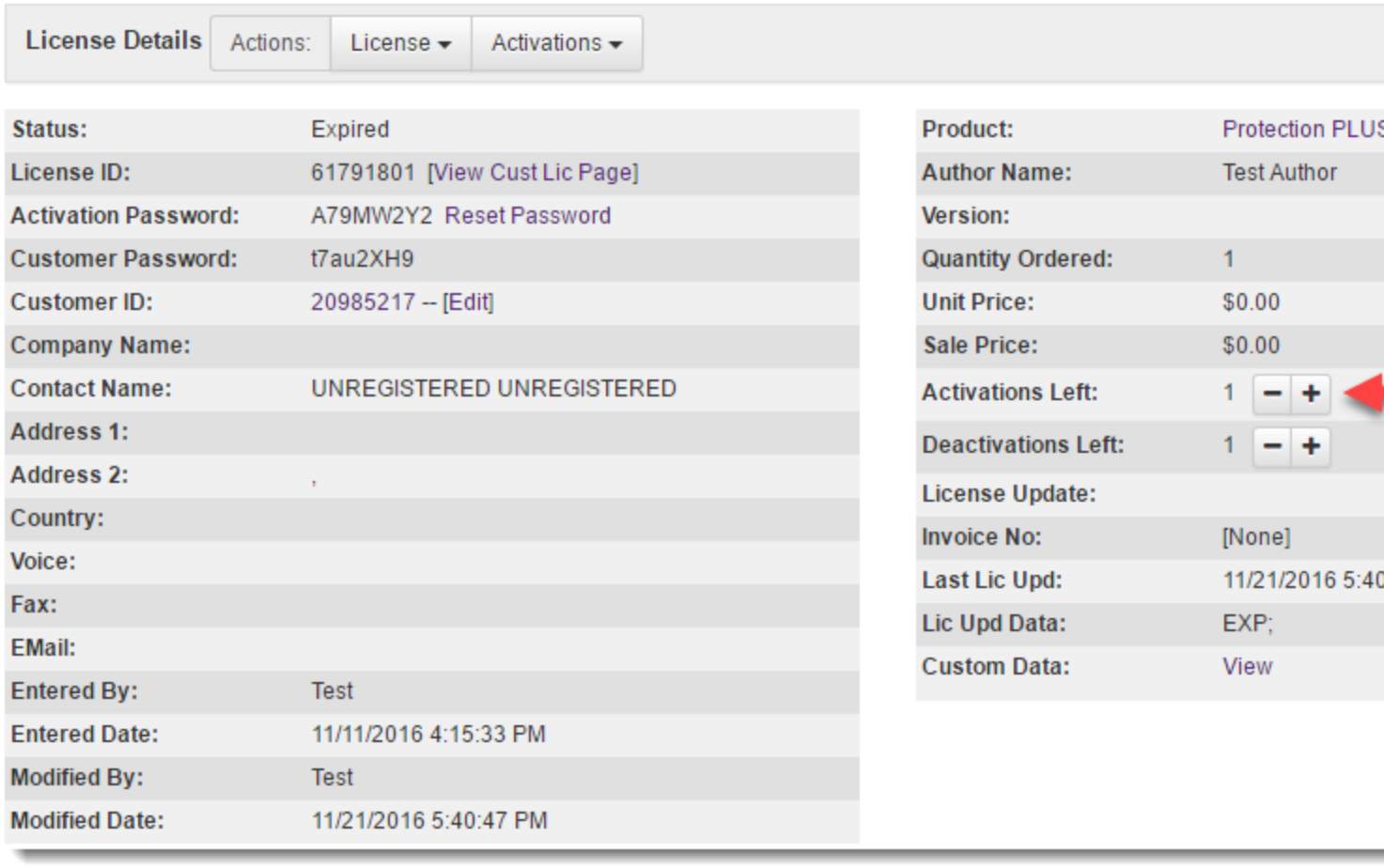


Click on the Administration Home breadcrumb and you will be shown the license requires an activation.



Step 4 - Process a Manual Activation

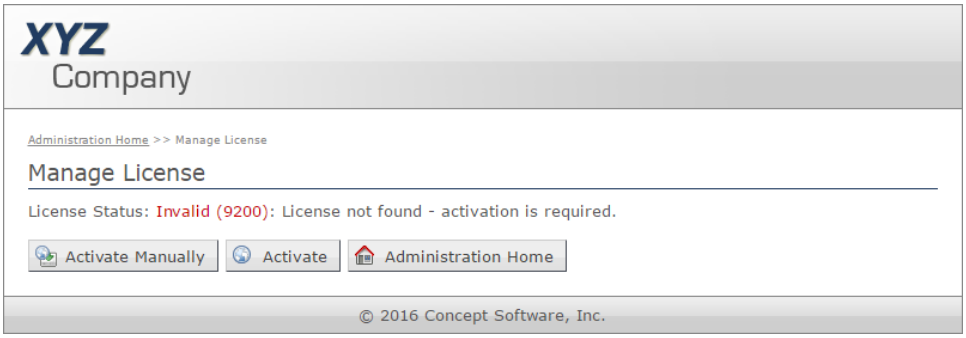
Before we can process another activation we need to set the status of our test license back to "OK" and increment the activations left as we used the one allocated activation already. You may skip resetting the license by following Step 2 again and creating a new test license.
Back in the License Details page in Instant SOLO Server click the + button to the right of the Activations Left field.



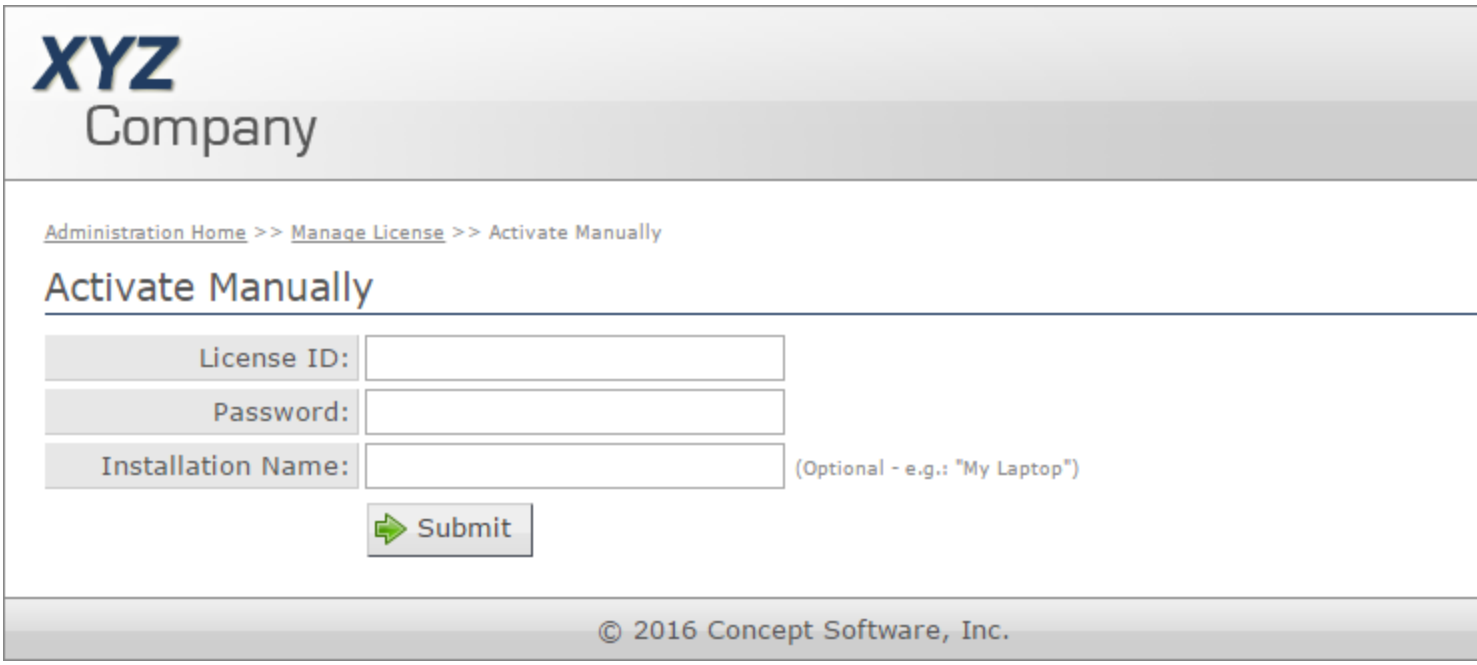
Click the Actions dropdown and choose *Edit*. Change the Status dropdown from Expired to OK then click Submit at the bottom.

The License Details page now should display a Status of "OK" and an Activations Left of 1.

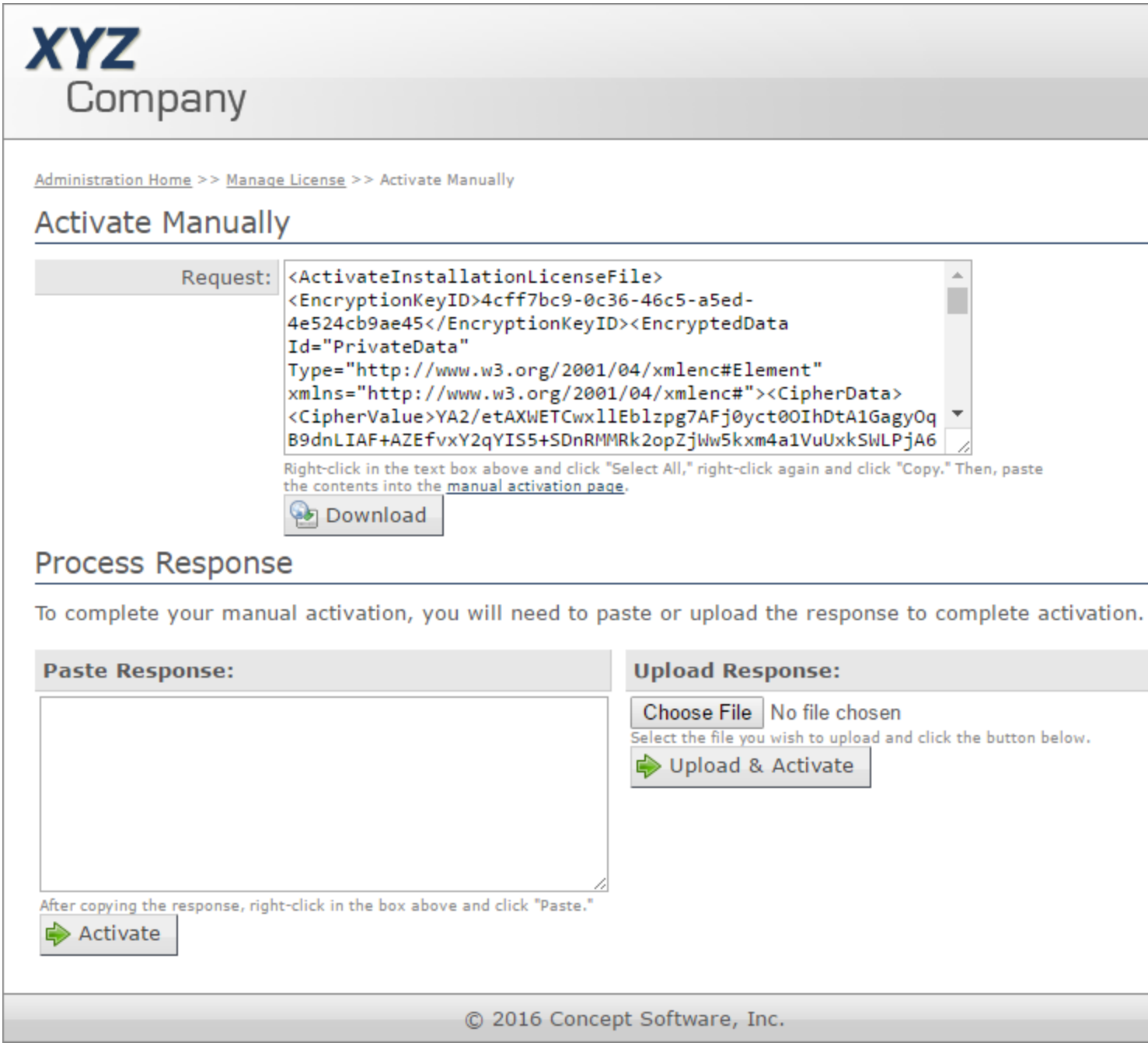
Back on the Manage License page of the sample click the Activate Manually button.



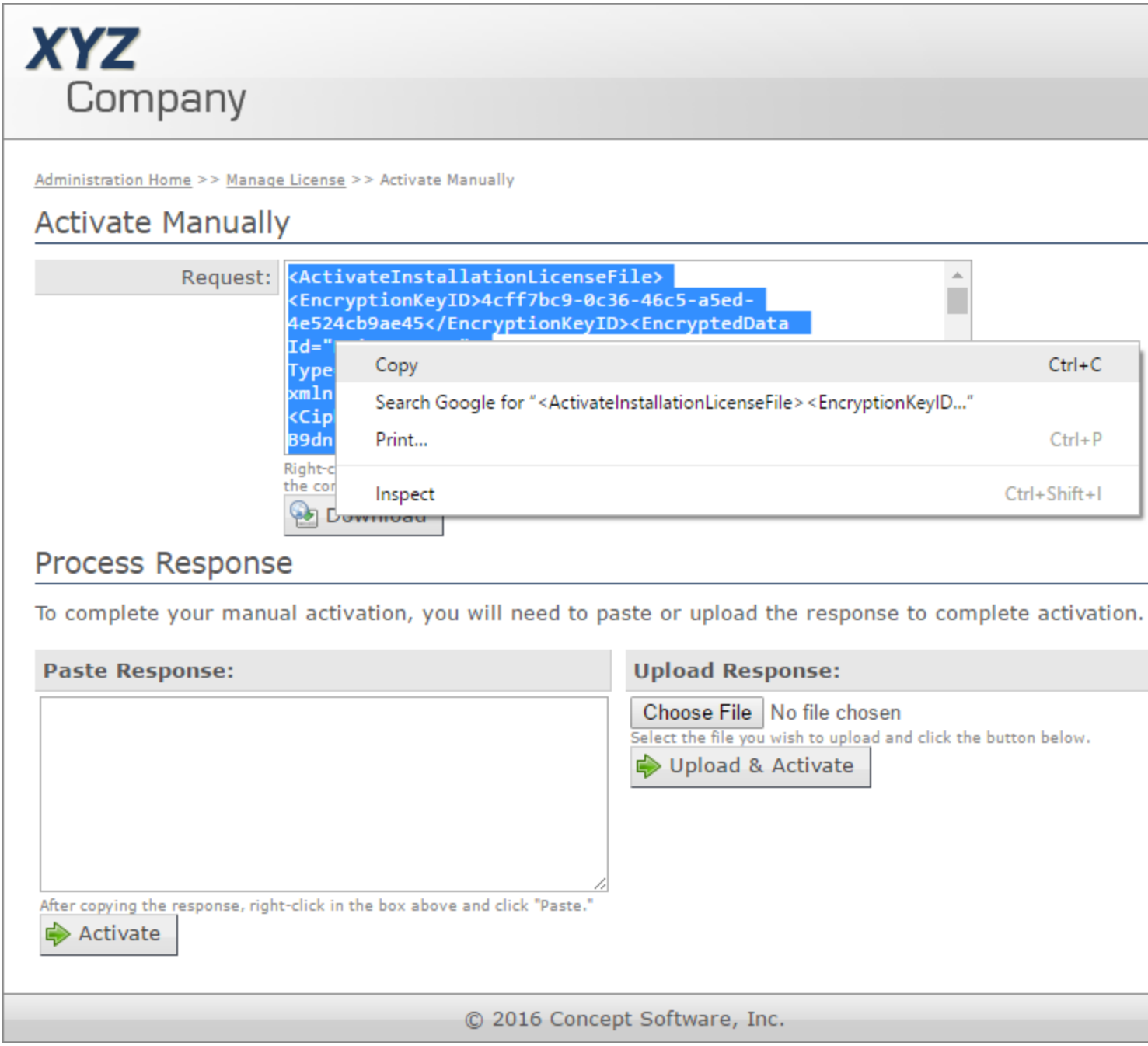
Once again enter the License ID and Password obtained from Step 2 above to generate the Activation Request.



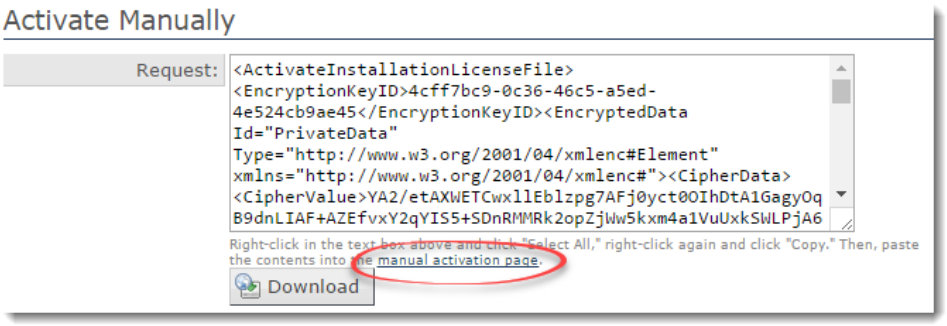
You will be shown the Activation Request page. In the Request text box will be an encrypted activation request to be processed by the activation server.



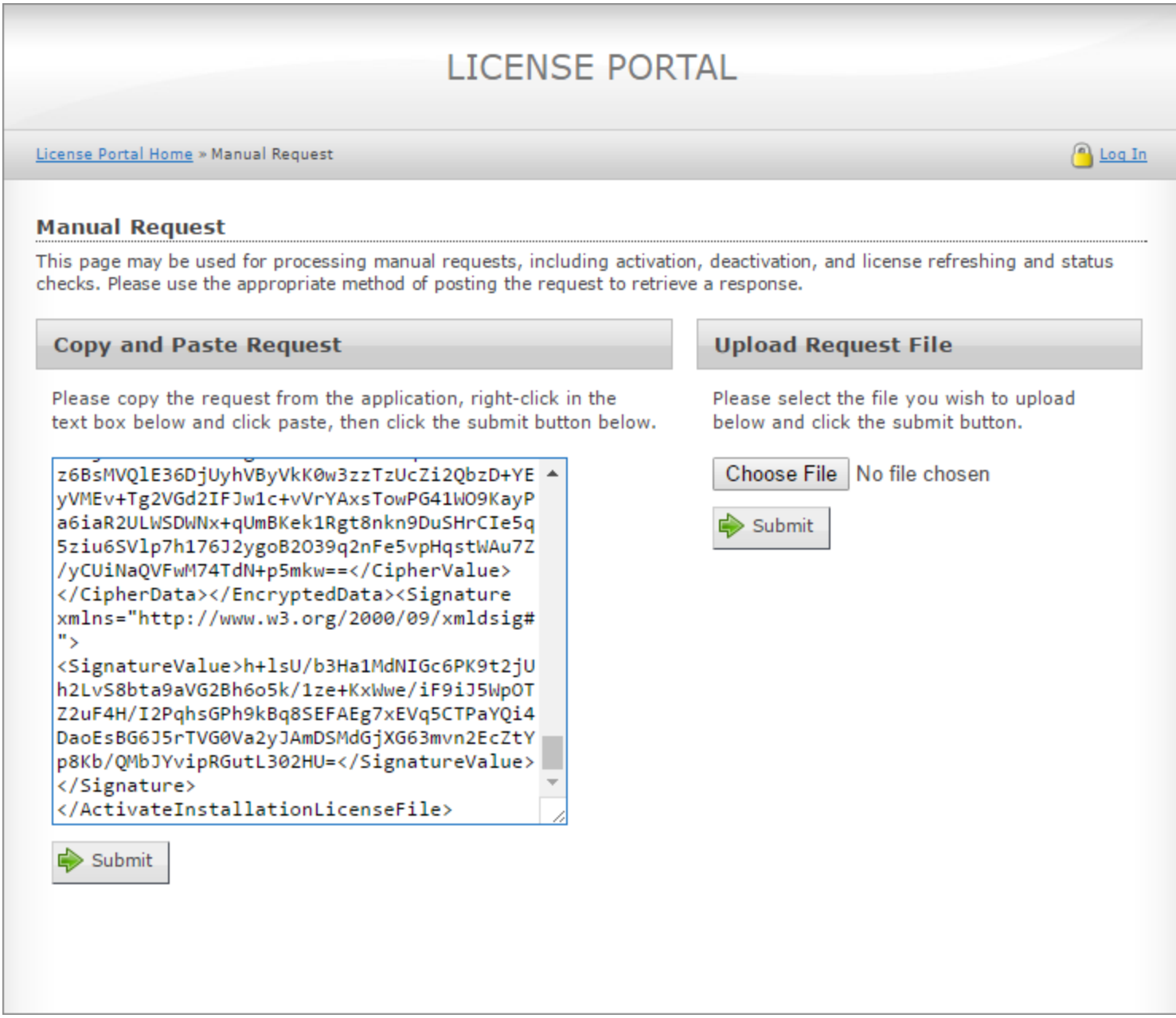
Right-click in the Request text box and click "Select All," right-click again and click "Copy" to copy the contents to your clipboard.



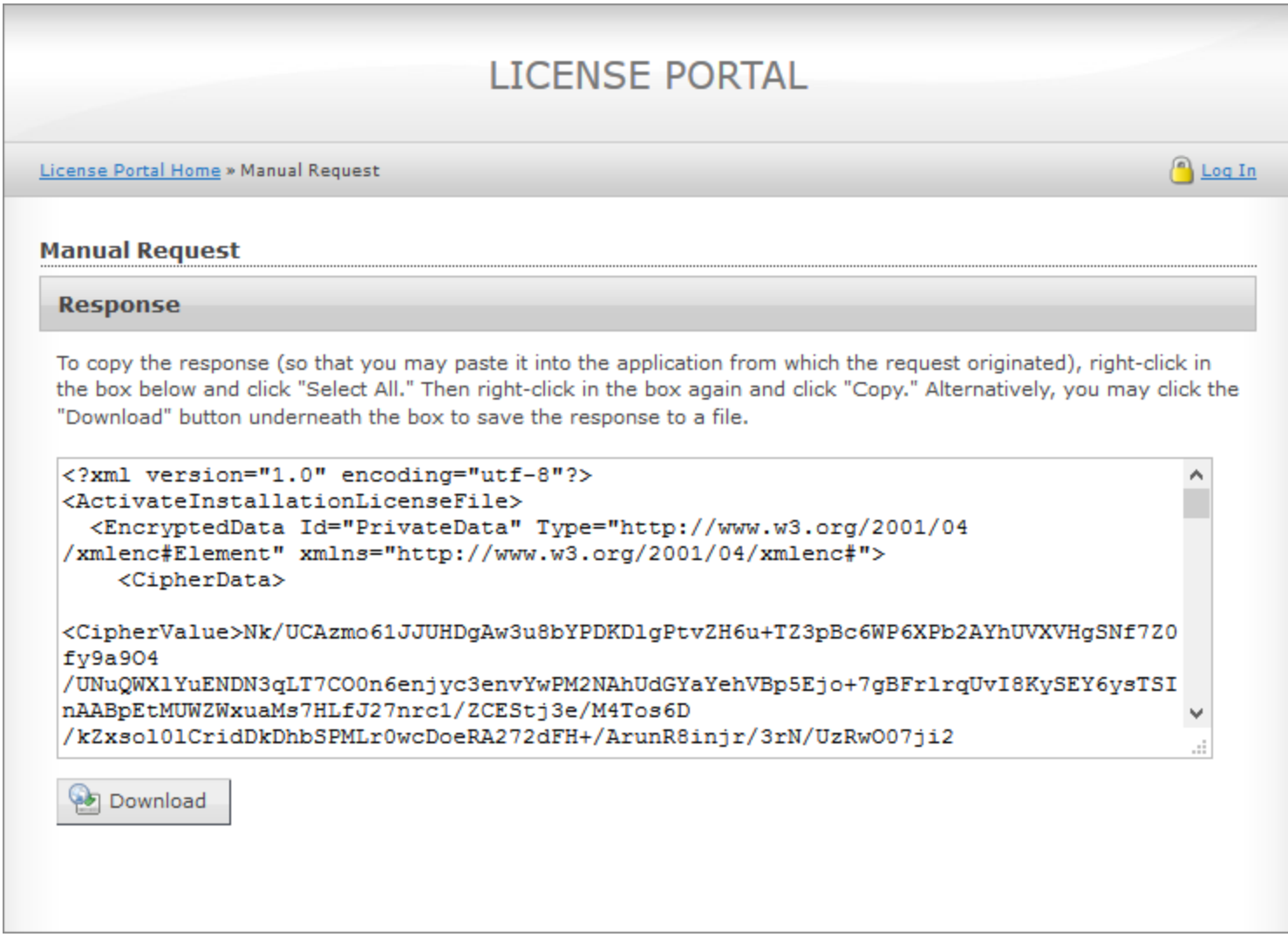
Click the "manual activation page" link on the page to be taken to the License Portal.



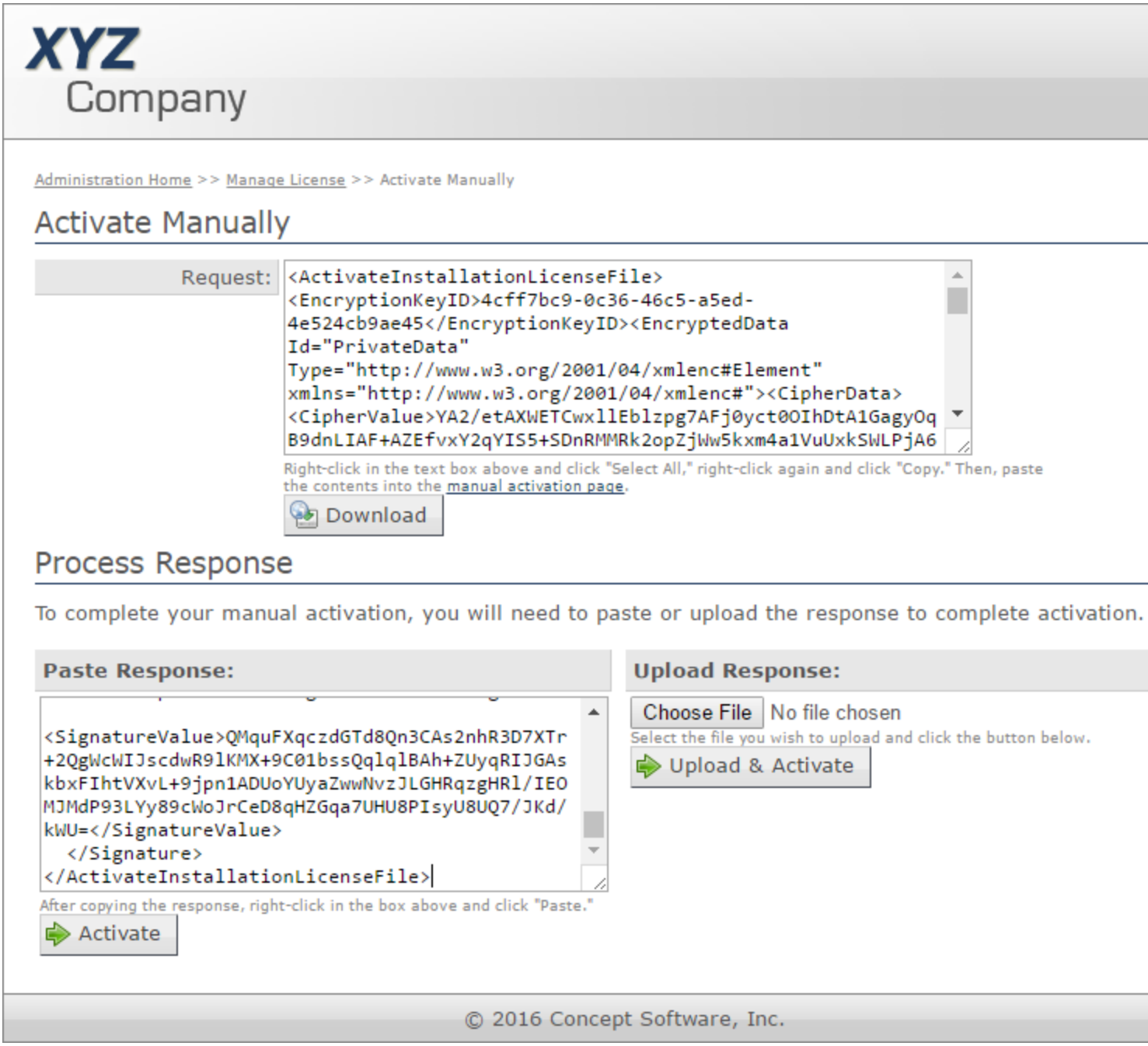
This web page is accessible from another computer with access to the Internet in the event that an offline workstation needed to be activated. Once you arrive to the License Portal page, paste the encrypted activation request data, and click Submit.



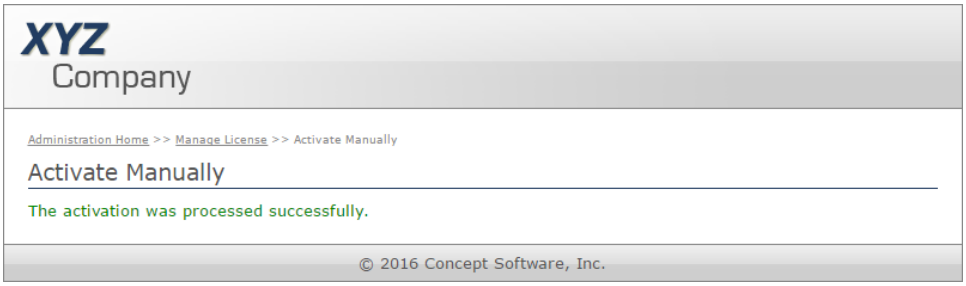
The License Portal will now generate an encrypted response. Copy this data to your clipboard.



Return to the Activate Manually page and paste the encrypted response into the Process Response text box and click Activate.



Upon successful activation, a confirmation is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



The License Status will now show as Fully Licensed.

PLUSManagedGui Sample .NET Applications

The PLUSManagedGui samples are similar to the **PLUSManaged samples** (which you should review if your application does not use a dialog/form based interface). However, in addition to integrating with **PLUSManaged**, these samples also **integrate with PLUSManagedGui**, which is a component that includes a graphical user interface (GUI) for licensing.

Overview

Here are some of the features and highlights to note about the PLUSManagedGui sample applications:

- The collection of samples included are each simply named after the type of license(s) supported. These are described in the Sample Projects section below.
- For the samples that use writable licenses (the "SelfSignedLicense" and SelfSignedTrial" samples), a 30-day evaluation is automatically started the first time the application runs on a computer.
- Activated licenses behave according to the type of license activated, which is defined by the Product Option (or the value in the TriggerCode property) in SOLO Server. This includes non-expiring licenses, time-limited (or lease) licenses, etc...
- These samples allow users to **activate** using a direct Internet connection, or manually using another device's Internet connection.
- Users can **deactivate** a previously activated license using a direct Internet connection, or manually via another device's Internet connection. This allows your application's users to essentially transfer the license to another computer. This is a convenient way to allow your customers to migrate to a new computer, without the need to contact you for support or additional activations. SOLO Server can limit the number of times a license is deactivated, and retains the history so you can make informed decisions when providing support.
- The application can **validate and refresh** its license with SOLO Server using a direct Internet connection, or manually using another device's Internet connection.
  - This can be done on demand, which is especially useful if you expect users may need to have a way to update their license properties quickly. (E.g. adding extra network seats, extending/renewing a time-limited license, etc...)
  - This can be done automatically with a direct Internet connection available, which is important so your application can detect when it has been deactivated previously or remotely.
  - You can also control how frequently your application tries to validate in this manner, and how frequently it is required to validate in this manner. This prevents network congestion, and helps avoid the perception that your application is slow or unresponsive should an unreliable network connection cause such delays. Further, when offering time-limited licenses that can be renewed/extended, this is a convenient way to automatically retrieve a new expiration date after the renewal has been processed.
- These samples also include basic support for proxy servers and proxy authentication.
- **Trigger code** activation is also supported, which allows you to activate customers when no Internet connection is available.
- **Volume licenses** are supported with these samples. These are special types of read-only licenses that are not authorized for any particular system.
- **Downloadable licenses with trigger code activation** are also supported with these samples. These are similar to volume licenses, except they are only authorized for a specific computer when activated using trigger codes. This gives you all the benefits of shipping a license file like you typically would when activated with SOLO Server, while not requiring an Internet connection to distribute or activate the license file. This can be convenient in scenarios where the protected application needs to be installed and activated on devices that are in an environment that lacks an Internet connection, or has very strict security.
- The NetworkFloatingSemaphore samples show how to limit the number of concurrent users (or instances of your application) that can use your protected application using exclusively locked semaphore files on a Windows (SMB) share.

Sample Source Files

Important

Many of the source code comments begin with `TODO` and `IMPORTANT`. **It is very important that you review each of these comments in any and all files you copy into your application.** Each of these comments indicates an area where you need to make an informed decision about how the application should behave and react in certain scenarios. Disregarding these comments can lead to undesirable behaviors in your application.

Sample Projects

The *%PUBLIC%\Documents\SoftwareKey\Protection PLUS 5\Samples* directory can be opened by running Protection PLUS 5 SDK from your start menu/screen, and clicking on the *Protection PLUS 5 SDK Samples* link. This directory contains all of the files and folders listed below, as well as several solution files (one for each supported version of Visual Studio). Each solution is configured with the sample projects that are designed for the corresponding version of Visual Studio, and the projects are organized by the underling framework and language used to develop each sample application. For example, any C# applications that use `System.Windows.Forms` dialogs are located in the *CSharp\Windows.Forms* solution folder (in the Solution Explorer pane/tab).

A summary of the samples included is provided below.

Name	Project Locations	Description
NetworkFloatingSemaphore	*CSharp\Windows.Forms\PLUSManagedGui_NetworkFloatingSemaphore* *VB.NET\Windows.Forms\PLUSManagedGui_NetworkFloatingSemaphore*	Shows how you can use semaphore files (or files locked to a running instance of your application) on a Windows (SMB) share to **limit the number of users or instances of your application** on a network.
ReadOnlyLicense	*CSharp\Windows.Forms\PLUSManagedGui_ReadOnlyLicense* *VB.NET\Windows.Forms\PLUSManagedGui_ReadOnlyLicense*	Shows how you can use a read-only **license file** for the highest level of **security**. These samples do not include evaluation licensing, though this is possible to add.
SelfSignedLicense	*CSharp\Windows.Forms\PLUSManagedGui_SelfSignedLicense* *VB.NET\Windows.Forms\PLUSManagedGui_SelfSignedLicense*	These samples always use a self-signed/writable **license file** for all licenses. These types of license files provide the highest level of flexibility, as they allow your application to freely modify the license file (even without Internet connectivity). However, using writable/self-signed licenses means the protected application uses key data fully known to it when encrypting license files, which is less **secure** than read-only licenses (which use key data only partially known to the protected application).

Source Code Files

The PLUSManagedGui samples use a variety of source code files that are common amongst many or all of the sample applications. The source files are summarized below, and each source file contains commenting which documents the source code in great detail. A summary of the shared source code files is provided below.

Class/Type	File Locations	Description
`MainForm`	*MainForm.cs*, *MainForm.Designer.cs*, *MainForm.vb*, *MainForm.Designer.vb*, *MainForm.resx* (Has other supporting files.)	Contains the main form or dialog implementation for the ReadOnlyLicense and SelfSignedLicense sample applications. This dialog is designed to provide a simple way of showing how your applications can interact with the licensing objects.
`LicenseConfiguration`	*LicenseConfiguration.cs* *LicenseConfiguration.vb*	Contains licensing configuration properties, including settings required for encryption, and settings which define how the application should behave and react in various circumstances. It is **very important** to review all of the `TODO` and `IMPORTANT` comments throughout this source file. This file contains code that requires changes before using it with your application.
`SampleLicense`	*SampleLicense.cs* *SampleLicense.vb* (Has other supporting files.)	This is a partial class which contains common definitions and methods used for working with and altering licenses. Additional, partial class files are included (such as *SampleReadOnlyLicense.cs* or *SampleSelfSignedLicense.vb*) to implement logic specific the corresponding license implementation class. Each sample application project only includes the partial class file relevant to the type of license implemented.
`SampleLicense`	*SampleReadOnlyLicense.cs* *SampleReadOnlyLicense.vb*	Implements the `License` class in PLUSManaged and provides an implementation that only uses read-only **license files**. This implementation includes support for **downloadable and volume licenses**.
`SampleLicense`	*SampleSelfSignedLicense.cs* *SampleSelfSignedLicense.vb*	Implements the `WritableLicense` class in PLUSManaged and provides an implementation that only uses writable/self-signed **license files**. Since **downloadable and volume licenses** may only be read-only, this relies on the `VolumeLicense` class to support these types of licenses.
`VolumeLicense`	*VolumeLicense.cs* *VolumeLicense.vb*	Implements support for **downloadable and volume licenses** for samples that use the

	`SelfSignedLicense` class for other types of licenses.

Configuring your licensing options

The `LicenseConfiguration` class contains a variety of settings, many of which are extremely important to change before re-using the sample source code in your applications. These settings are important for security, and many of them control the application's licensing behavior. As noted earlier, many of the source code comments in this file begin with `TODO` and `TODO: IMPORTANT`. These comments indicate an area of code or a setting that you need to review and, in many cases, update. These settings are outlined in the table below.

Setting Name(s)	Description
Encryption Settings	
`EncryptionKey`	This contains the encryption key data, which usually comes from **your SOLO Server account**. **It is very important that you update this to use your account's data before distributing your application!** If you are evaluating Protection PLUS 5 SDK, also make sure you update the Encryption Key to a non-expiring key which will no longer have the "_EVALUATION_EXPIRES_2013-10-05_" expiration date at the beginning (the actual date will be different). **If you distribute your application with an evaluation key, it will expire and your customers won't be able to use the software.** If you have purchased a license for Protection PLUS 5 SDK but your Encryption Key still has an expiration date, please **contact us**. Additionally, it is important to use the correct key store in this property. In most cases, your desktop applications should use the user key store, while other types of applications, such as services and web applications/services, should typically use the machine key store.
`ManualActionIV`, `ManualActionKey`	When using PLUSManagedGui, you can configure the component (at design time or run time) to allow one or more actions (such as activation, deactivation, and refresh) to be done manually. This enables your customers to leverage

	another device's Internet connection to manually activate a system which is not connected to the Internet. When using this functionality, it is important to encrypt the data being used in the requests when saving it for later use (so the user can close the application, and complete the request later), and these properties serve as the key used to encrypt this data. When reviewing the `LicenseConfiguration` class, the source code contains comments above this property that includes sample code which shows how to generate a new key. Each application you protect should have its own, unique key for this property.
`RegKey2Seed`, `TriggerCodeSeed`	When adding support for trigger codes to your application, it is very important that these values are updated to contain values that are unique to each application you protect. When using SOLO Server, these values must also match the corresponding values in your Product Option(s) that are configured for the given application. The RegKey2Seed must contain a value between 1 and 255, while the TriggerCodeSeed must contain a value between 1 and 65535.
Application & Product Settings	
Application Directory	Gets the absolute path to the directory in which the application (executable file) is located. This property is present as a convenience, and is used by other properties when storing other files (such as the license file, aliases, etc...) in the same directory as the sample application.
ThisProductID	Gets the application's Product ID. This value is typically issued when **configuring a product in SOLO Server**. However, if you are not using SOLO Server, this should contain a value that is unique to each application you protect.
ThisProductVersion	Gets the application's version number. By default, this uses the assembly version of

	the protected application. This value must contain 4 parts, no longer than 5-digits each (e.g. N.N.N.N, where each N is at least 0, and no larger than 99999).
License File & Alias Settings	
PATH_REGISTRY_LOCATION	This constant specifies the Windows registry key value (stored under HKEY_CURRENT_USER) that will be used to store the license file path specified with the NetworkFloatingSemaphore samples. When implementing network floating, it is very important to change this key location to something that is unique to each application you protect.
Aliases	Gets a list of aliases used with any samples that use writable license files. By default, this stores the aliases in the same directory as the application or license file for convenience. (This makes it easier for you to simulate running your application on a computer for the first time since its easier to delete the license file and aliases this way.) However, it is very important that you pick better locations to hide/obscure these files for your protected applications. Additionally, it is also very important to make sure the locations selected are unique to each application you protect.
LicenseFilePath	Gets the path to the application's license file. By default, this file is stored in the application directory for convenience.
ManualActionSessionStatePath	When using PLUSManagedGui, you can configure the component (at design time or run time) to allow one or more actions (such as activation, deactivation, and refresh) to be done manually. This enables your customers to leverage another device's Internet connection to manually activate a system which is not connected to the Internet. This property gets the path to the manual action session state file, which is what stores a manual request so it can be completed at a later time.

PathRegistryValue	When using the NetworkFloatingSemaphore samples, this gets or sets the registry value that contains the path to the license file and semaphore files. The registry location is defined in the PATH_REGISTRY_LOCATION constant described above.
NetworkSemaphorePrefix	When using the NetworkFloatingSemaphore samples, this sets the prefix used for the semaphore file names. So for example, with the default prefix of "sema", the semaphore files created will be named like "sema123.net".
Licensing Restrictions & Settings	
FreshEvaluationDuration	When using samples that use a writable license file, this specifies the number of days in which a new evaluation period will last.
RefreshLicenseAlwaysRequired	Specifies whether or not the protected application will attempt to validate and refresh its license with SOLO Server every time it starts or validates the license.
RefreshLicenseAttemptFrequency	Specifies the number of days to wait before attempting to validate and refresh the license with SOLO Server.
RefreshLicenseEnabled	This property is present for convenience, and evaluates RefreshLicenseAlwaysRequired, RefreshLicenseAttemptFrequency, and RefreshLicenseRequireFrequency to determine whether or not license refreshes are enabled at all.
RefreshLicenseRequireFrequency	Specifies the number of days to wait before requiring the protected application to validate and refresh its license with SOLO Server.
RuntimeBackdateThresholdSeconds	The amount of time (in seconds) to allow the system clock to be back-dated during run-time.

SystemIdentifierAlgorithms	The system identifier algorithms to use to uniquely identify and authorize a system. The identifiers generated by these algorithms are pivotal for **adding copy protection** to your applications.
TimeLimitedWarningDays	When using an activated time-limited license in the PLUSManagedGui samples, this specifies the number of days to begin warning the user that the license will soon expire.
Volume & Downloadable License File Settings	
DownloadableLicenseOverwriteWithNewerAllowed	When using downloadable licenses with trigger code activation, this setting determines whether or not your users can update their license file with a newer file from SOLO Server. Allowing this could make it easier to let your customers update their license through SOLO Server's customer portal after renewing a subscription, or updating registration data, for example.
DownloadableLicenseOverwriteWithOlderAllowed	When using downloadable licenses with trigger code activation, this setting determines whether or not your users can update their license file with a older file. While this feature is not typically needed, we do recommend requiring activation when allowing this.
DownloadableLicenseOverwriteWithNewerRequiresActivation	When using downloadable licenses with trigger code activation, this setting determines whether or not your protected application will require the user to process a trigger code activation to use a new license file.
DownloadableLicenseOverwriteWithOlderRequiresActivation	When using downloadable licenses with trigger code activation, this setting determines whether or not your protected application will require the user to process a trigger code activation to use an older license file.
VolumeLicenseFilePath	In PLUSManagedGui samples that use self-signed (writable) license files, this

	specifies the path used for volume license files (which are always read-only).
`VolumeSystemIdentifierAlgorithms`	This specifies the system identifier algorithms used for volume licenses which is generally the LicenseIDIdentifierAlgorithm (as the volume license is not authorized for any particular system, by definition).

PLUSNative Sample Applications

The PLUSNative samples show you how you can **integrate the PLUSNative API** with your applications. These samples include support for a variety of popular languages and GUI frameworks, including (but not limited to) C++ with MFC, C++ with wxWidgets, Delphi, Objective C, and Visual Basic 6.

Include Files

The Protection PLUS 5 SDK installation directory has a sub-directory named *Include*. Each file in this directory contains definitions your application needs to call PLUSNative, and in many cases, it may also contain helper functions that simplify the use of PLUSNative by addressing language-specific limitations and nuances. (An example if this is how the Visual Basic 6 PLUSNative.bas file has its own implementation for `SK_ApiResultCodeToString`, so you do not need to worry about how to properly deal with string buffers allocated by our library in your code.)

Library Files

The Protection PLUS 5 SDK installation directory also contains a sub-directory named *Library*. Here, you will find a variety of builds of the PLUSNative API, which are organized as outlined below.

Dynamic Link Libraries / Shared Objects

Dynamic Link Libraries (DLLs), or shared objects, are organized with the following structure: *Operating System/[DLL/shared]/Architecture*. So for example, the 64 bit Windows DLL is located in *Library\Windows\DLL\x64*, and the universal OS X library is located in *Library/macOS/shared/universal*. When used, you will need to **distribute** one or more library files with your protected application(s).

Static Libraries

Static libraries allow you use PLUSNative with your application(s) without the need to **distribute** additional library files (this only applies to languages that can statically link a C library, such as C/C++). These files are organized with the following structure: *Operating System/Development Environment/[import/static/static-nodeps]/Architecture*. The third-level of this structure describes the nature of the static libraries, which include:

- import: These are import libraries that may be linked when using Dynamic Link Libraries (DLLs)/Shared Objects.
- static: These are static libraries which include third-party dependencies used by PLUSNative, such as OpenSSL, libcurl, and libxml2.
- static-nodeps: These are static libraries which do not include third-party dependencies used by PLUSNative. There are some cases (unrelated to the use of PLUSNative) where you might need to link your own dependencies, such as if you need to use a different version, or if you use a custom patch to resolve some issue your application experienced with the dependency. For these niche scenarios, the libraries in static-nodeps may be used, but leave you responsible for linking all of the required dependencies (OpenSSL, libcurl, and libxml2).

Next, you might also notice the static libraries for Windows include two versions named PLUSNative.lib and PLUSNative_md.lib. This is related to the *Runtime Library* setting in Visual C++ projects (under *Configuration Properties/C++/Code Generation* in the project properties). The PLUSNative.lib file may be used with the "Multi-threaded (/MT)" and "Multi-threaded Debug (/MTd)" runtime libraries, while the PLUSNative_md.lib file may be used with the "Multi-threaded DLL (/MD)" and "Multi-threaded Debug DLL (/MDd)" runtime libraries.

Sample Source Files

On Windows, the *%PUBLIC%\Documents\SoftwareKey\Protection PLUS 5\Samples* directory can be opened by running Protection PLUS 5 SDK from your start menu/screen, and clicking on the *Protection PLUS 5 SDK Samples* link. On other operating systems, you will choose where to extract the development files during installation, which will include the samples.

The sample directory contains sub-directories organized in the following manner: *Operating System/Development Environment/Language/GUI Framework/Sample Project Directory*. So for example, the MFC samples can be found under the *Windows\Visual Studio\C++\MFC* sub-directory, and Objective C samples may be found under the *macOS/Xcode/Objective-C/Cocoa* subdirectory.

Important

Many of the source code comments begin with `TODO` and `IMPORTANT`. **It is very important that you review each of these comments in any and all files you copy into your application.** Each of these comments indicates an area where you need to make an informed decision about how the application should behave and react in certain scenarios. Disregarding these comments can lead to undesirable behaviors in your application.

The "LicensingSample" Sample Projects

Any "LicensingSample" project, regardless of the language and GUI framework used in the individual sample, supports the same set of licensing features out-of-the-box. This includes features such as online activation, manual activation, copy-protection, and more. Additionally, these samples support different types of licenses, including time-limited evaluation/trial licenses, perpetual (non-expiring) licenses, and activated time-limited (or lease/periodic) licenses. The sample is also smart enough to automatically use a read-only **license file** issued by SOLO Server when appropriate (such as for perpetual licenses), or use writable license files when appropriate (such as for time-limited licenses, including evaluation licenses).

The names and contents of the files included with the LicensingSample projects do vary based on language and GUI framework; however, each version of this sample project does have the same general set of items:

- The main form or dialog (usually named something like MainForm, MainDlg, etc...), which represents what would instead be your application's main form/dialog after integration. This can include buttons or menu options that allow the end-user to check the status of and manage his or her license.
- The online activation form (which generally has a name that starts with something like ActivateOnline) is used to prompt users for a License ID and password needed to activate using SOLO Server.
- The manual activation form (which generally has a name that starts with something like ActivateManually) allows the user to generate an activation request that can be sent to SOLO Server using a different device (which has Internet connectivity), and allows the user to then complete activation after bringing the response back to the disconnected device.
- A License class (usually named License, or SKLicense) that contains most of the application's licensing functionality. This contains the licensing code that you could reference or copy when integrating PLUSNative into your application.

Of course, some versions of the project could include additional source files, such as customized controls (like text boxes that limit user input).

LicensingSample Features

The LicensingSample projects all contain the same general features and functionality, regardless of the target language or platform. The most relevant of the features are summarized below.

- When the sample is first run, and no prior license exists, it automatically creates a writable license file that is valid for a 30 day evaluation of the sample application. Of course, this behavior can be altered or omitted for your protected applications.
- These samples allow users to **activate** using a direct Internet connection, or manually using another device's Internet connection.
  - When a time-limited license is activated, it is saved to a writable license file, which can help prevent system back-dating.
  - When a perpetual/non-expiring license is activated, it is saved as the original read-only license file issued by SOLO Server.

- Activation may be completed using a direct Internet connection, or may be completed manually by using another device's Internet connection.
- Once activated, a license may be refreshed or deactivated using a direct Internet connection on the licensed system.
- Using a direct Internet connection, users can **deactivate** a previously activated license, which allows your application's users to essentially transfer the license to another computer. This is a convenient way to allow your customers to migrate to a new computer, without the need to contact you for support or additional activations. SOLO Server can limit the number of times a license is deactivated, and retains the history so you can make informed decisions when providing support.
- The application can **validate and refresh** its license with SOLO Server using a direct Internet connection.
  - This can be done on demand, which is especially useful if you expect users may need to have a way to update their license properties quickly. (E.g. adding extra network seats, extending/renewing a time-limited license, etc...)
  - This can be done automatically, which is important so that your application can detect when it has been deactivated previously or remotely.
  - You can also control how frequently your application tries to validate in this manner, and how frequently it is required to validate in this manner. This prevents network congestion, and helps avoid the perception that your application is slow or unresponsive should an unreliable network connection cause such delays. When offering time-limited licenses that can be renewed/extended, this also is a convenient way to automatically retrieve a new expiration date after renewal.
- These samples include the source code for all forms/dialogs used, which makes it easy for you to customize in any way needed.

Making the LicensingSample Work Exclusively with Read-Only Licenses

The LicensingSample demonstrates using read-only licenses and falling back to writable licenses. For those using only read-only licenses there are a few code changes that must be made to the sample code. The License class contains a `Reload` method which contains the few lines of code needed to be removed and modified in order to have the LicenseSample only use read-only licenses. Normally, in this method, if the read-only license is not found the code falls back to open a writable license and if that is unsuccessful it will create a writable license. The new method should remove the fallback code and look like the following code snippet.

C/C++

```
void License::Reload()
 {
   SK_ResultCode result = SK_ERROR_NONE;
    m_IsLoaded = false;
   SetWritable(false);
   // Try to load the license file as read-only
   if (SK_ERROR_NONE != (result = SK_PLUS_LicenseFileLoad(m_Context, SK_FLAGS_NONE, ToUTF8(m_
   LicenseFilePath).c_str())))
    {
       m_LastErrorNo = result;
    }
   else
    {
       m_IsLoaded = true;
    }
   // Notify our application to update it's status
   if (m_RefreshLicenseStatusCallback)
      m_RefreshLicenseStatusCallback();
 }
```

In addition to the above changes, you will need to make changes in the `RefreshLicenseStatus` method callback of the `MainDlg` by removing the evaluation check code from the method. As there may not be a license file loaded, you will need to move the code to read and append the registration data into the `if (licenseValid)` code block.

The "CloudContolledNetworkFloatingSOLOServer" Sample Projects

These projects show how you can leverage PLUSNative and SOLO Server to provide **Cloud-Controlled Floating Network Licensing using SOLO Server**. For more information, you can read more about:

- **Cloud-Controlled Floating Network Licensing via SOLO Server**
- **Cloud-Controlled Floating Network Licensing via SOLO Server using PLUSManaged** (for .NET applications)
- **Cloud-Controlled Floating Network Licensing via SOLO Server using PLUSNative** (for native applications)

Note that this is presently a preview of the feature. If you wish to use Cloud-Controlled Floating Network Licensing using SOLO Server, you will need to **contact us** for additional details on how to do so.

Configuring your system to use the wxWidgets samples

Although PLUSNative includes samples that use wxWidgets, it is **not** a dependency or requirement for using PLUSNative. However, it is one of many GUI libraries that is convenient to use because of its support for multiple operating systems. The PLUSNative samples require wxWidgets 2.8 or later. If you are downloading **wxWidgets** manually, then we generally recommend using the latest version to maximize compatibility with the latest versions of operating systems. Also, be careful to download the version with the DOS line endings when building for Windows, and download the version with Unix line endings when building for other operating systems.

The directions below are designed to help you get started with the wxWidgets samples that ship with PLUSNative, but they do not provide additional instruction for making multiple builds of wxWidgets for different processor architectures (x86, x64) or different underlying GUI libraries. This means that, if your application needs to support multiple processor architectures, and/or multiple GUI libraries, you are responsible for establishing the additional steps and project/build configurations as necessary.

Download wxWidgets using NuGet on Windows

If you use Microsoft Visual Studio to build your Windows applications (or if you are just using it to review our sample applications), the easiest way to get started with the wxWidgets samples is to use the wxWidgets NuGet package. Simply follow the instructions at **https://wiki.wxwidgets.org/Microsoft_Visual_C++_NuGet** to download the templates and configure the project.

Manually building wxWidgets on Windows

Step 1: Determine the version of the Visual C++ compiler you will be using

If you only have one version of Visual Studio installed, then you will be using that version to compile your C++ applications. However, it is also possible to use a newer version of the Visual Studio IDE while using an older version of the platform toolset to use an older version of the C Runtime (CRT). (This is often done to support older versions of Windows, while still leveraging newer IDE features.) To determine the version of the Visual C++ compiler your project uses, open the project and click the Project/Properties menu, and observe the value selected under the "Platform Toolset" property on the Configuration Properties/General page. The value listed here will also tell you the version number (e.g. "Visual Studio 2008 (v90)"), which you will need in a later step.

Step 2: Determine which CRT is being used

The C Runtime (CRT) can be linked dynamically or statically, and it is important that you compile wxWidgets to link the CRT the same way your application does. (Most applications will link the CRT dynamically.) You can view this setting by opening your project, click the Project/Properties menu, expanding the C/C++ section, selecting the "Code Generation" page, and observing the value in the "Runtime Library" property. If this property is set to "Multithreaded DLL (/MD)" or "Multithreaded Debug DLL (/MDd)", then your program links the CRT dynamically. Otherwise, if this is set to "Multithreaded (/MT)" or "Multithreaded Debug (/MTd)", then your program links the CRT statically. You will need to take note of this for a later step.

Step 3: Determine the target architecture

If you open your project, click the Project/Properties menu, you will see a "Platform" option that shows what architecture you are targeting. If it shows Win32, then your application should be targeting 32-bit (x86) processor architecture. If it shows x64, then it is targeting 64 bit (x64, also known as amd64) processor architecture. If you expand the Linker section, click Advanced, and observe the "Target Machine" property, you can also confirm the target architecture here.

Step 4: Build wxWidgets

Now that you have gathered all the required information, you can finally build wxWidgets using a command similar to the one below:

Windows Command Prompt

```
"%VS110COMNTOOLS%..\..\VC\bin\amd64\vcvars.bat"
nmake /f makefile.vc BUILD=release DEBUG_INFO=0 RUNTIME_LIBS=static TARGET_CPU=X64
```

You may need to alter the highlighted parts of the commands shown above as follows:

1. `%VS110COMNTOOLS%` is an environment variable that points to the Visual Studio 2012 directory. If you determined your application targets a C Runtime library for a different version of Visual C++, then you will need to update this to use the appropriate environment variable for that version. Additionally, the name of "vcvars.bat" may vary depending on the architecture being targeted and the version of Visual Studio being used. You may need to browse to the directory where this batch file is located to determine the name of the file.
2. If you determined that your application is linking the C Runtime library dynamically in step 2, then remove `RUNTIME_LIBS=static` from the command.
3. If you determined that your application is targeting 32 bit (x86) architecture, then remove `amd64\` from the first line, and remove `TARGET_CPU=X64` from the second line.

Step 5: Set your environment variable

Most wxWidgets applications, and consequently the PLUSNative sample applications, use a `WXWIN` environment variable to determine where the wxWidgets header files and libraries may be found. You will need to add this environment variable, and set its value to the directory where you have just built wxWidgets. Once this is done, the PLUSNative samples should compile and link. If you run into difficulty getting them to do so, you may contact us to **get help**.

Manually building wxWidgets on macOS

If you already use a third-party package management system (such as Fink, Homebrew, or MacPorts) on your development system, it is possible to leverage this to install wxWidgets. However, this might not suffice for your needs if you need to statically link wxWidgets, or distribute the libraries with your application. Furthermore, it is also important that you use the latest version of wxWidgets, as version 2.8 does not support 64 bit Cocoa applications.

To build wxWidgets, open a terminal, change directory to where you download and extracted wxWidgets, and run the commands shown below. The `--disable-shared` option makes it compile static libraries, and you can remove this if you prefer to compile shared objects.

Terminal

```
./configure --with-osx_cocoa --disable-shared
make
sudo make install
sudo ln -s /usr/local/bin/wx-config /usr/bin/wx-config
```

The last command creates a symbolic link to the wx-config script in your /usr/bin directory, as this is often the easiest way to allow the compiler to find this script. Our wxWidgets samples rely on this script to configure compiler and linker flags needed to use wxWidgets.

Also note that you may need to consider the minimum version of OS X you plan to target with your application. For example, our wxWidgets samples use the -mmacosx-version-min=10.5 flag to tell the compiler to make sure the sample application is compatible with OS X 10.5 and later. You may need to specify matchign flags when compiling wxWidgets to avoid linker errors.

Manually building wxWidgets on Linux

Linux distributions generally include a package management system which you can use to install wxWidgets and its development files. However, this might not suffice for your needs if you need to statically link wxWidgets, or distribute the libraries with your application. Additionally, you may need to use the latest version of wxWidgets, as version 2.8 does not support GTK+ 3.

Step 1: Install dependencies

Before you get started, you will need to determine what dependencies you will have, and ensure they are installed. For example, this often includes libgtk2.0-dev or libgtk-3-dev (for GTK+ 2 and GTK+ 3, respectively). Refer to the documentation for the Linux distribution you have installed for more information on how to install the required packages using its package management system.

Step 2: Build wxWidgets

To build wxWidgets, open a terminal, change directory to where you download and extracted wxWidgets, and run the commands shown below. Note that if your application uses GTK+ 2, you will need to change `--with-gtk=3` to `--with-gtk=2`. Additionally, the `--disable-shared` option makes it compile static libraries, and you can remove this if you prefer to compile shared objects.

Terminal

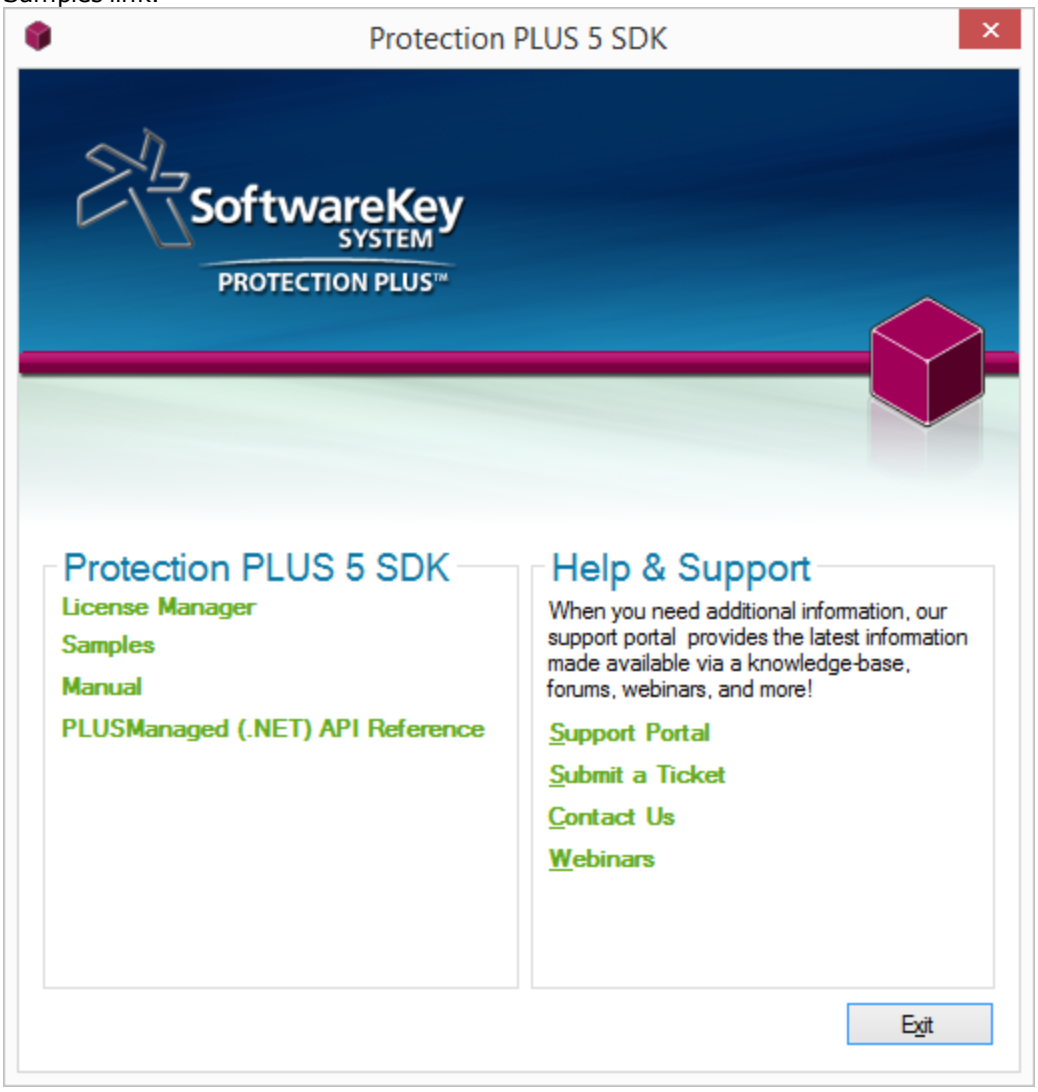
```
./configure --with-gtk=3 --disable-shared
make
sudo make install
```

Also note that not all Linux distributions use `sudo` for privilege escalation. If you are running such a distribution, you may need to run the `su` command before running the `make install` command.

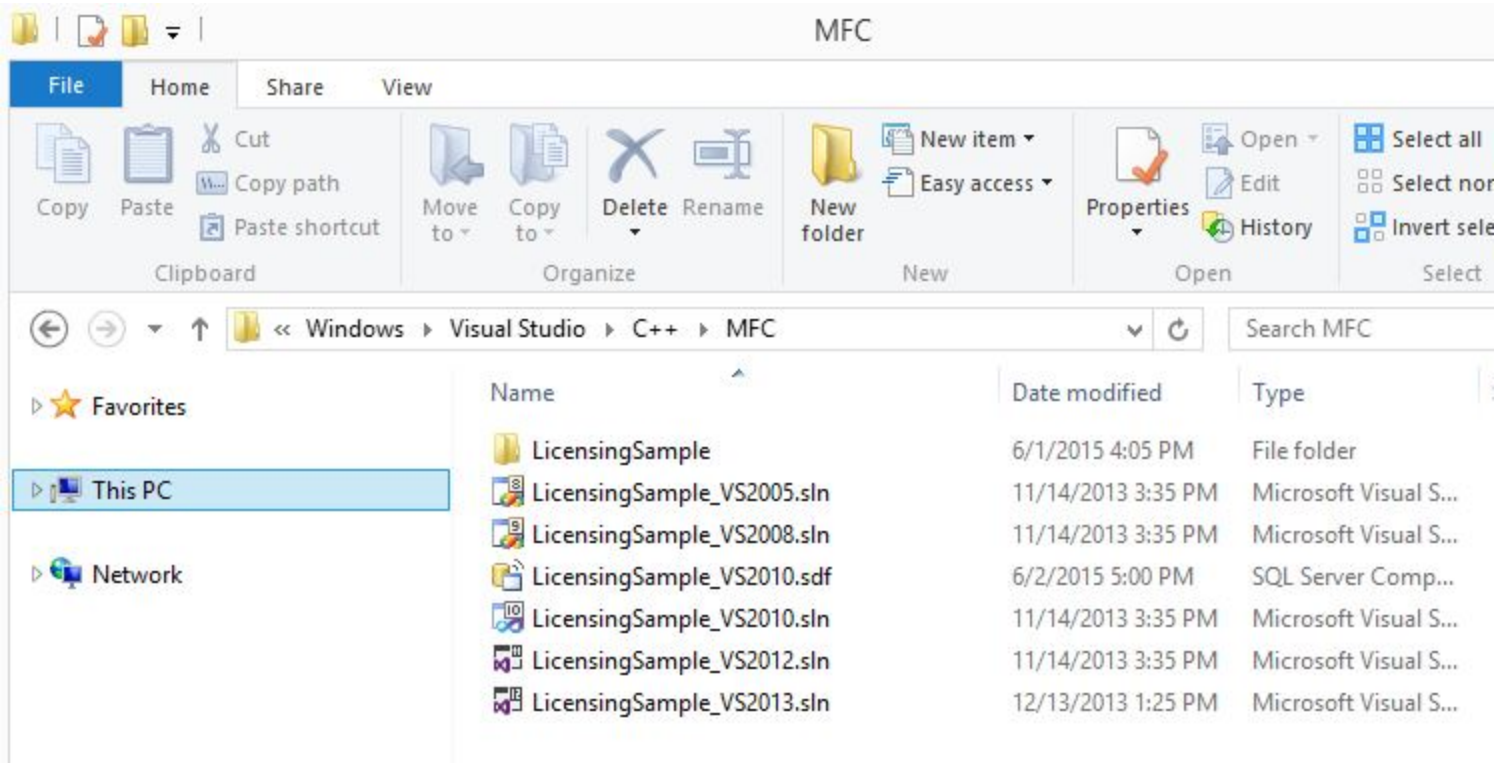
PLUSNative LicensingSample

Step 1 – Opening the Licensing Sample Project

Open the LicensingSample project file corresponding to your programming language. The Samples directory can be opened by running Protection PLUS 5 SDK from your start menu/screen, and clicking on the Protection PLUS 5 SDK Samples link.



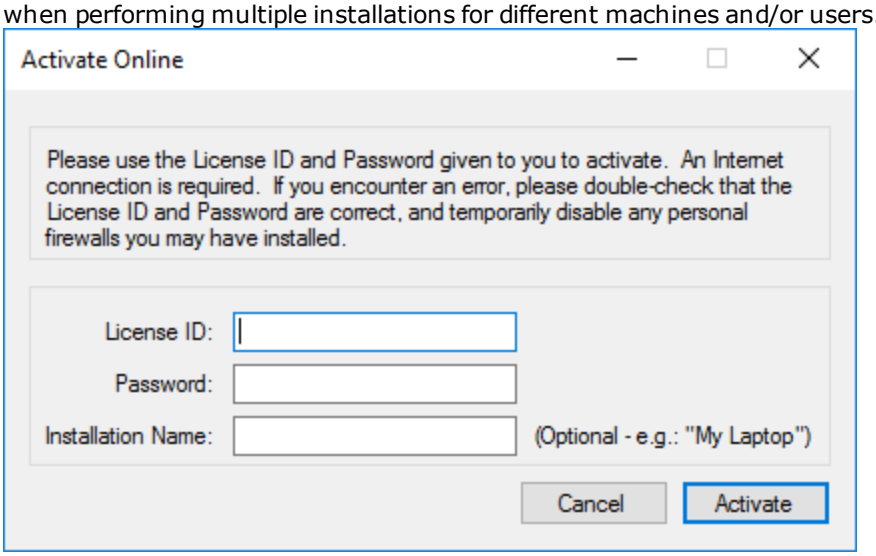
For this example, we will be using the MFC (C++) Sample in Visual Studio.



Right click the LicensingSample project and select Debug > Start New Instance.
img src="Resources/Images/ScreenShot/a03_PP5_LicensingSample_Debug.JPG" />

Step 2 – Activating automatically with Instant SOLO Server

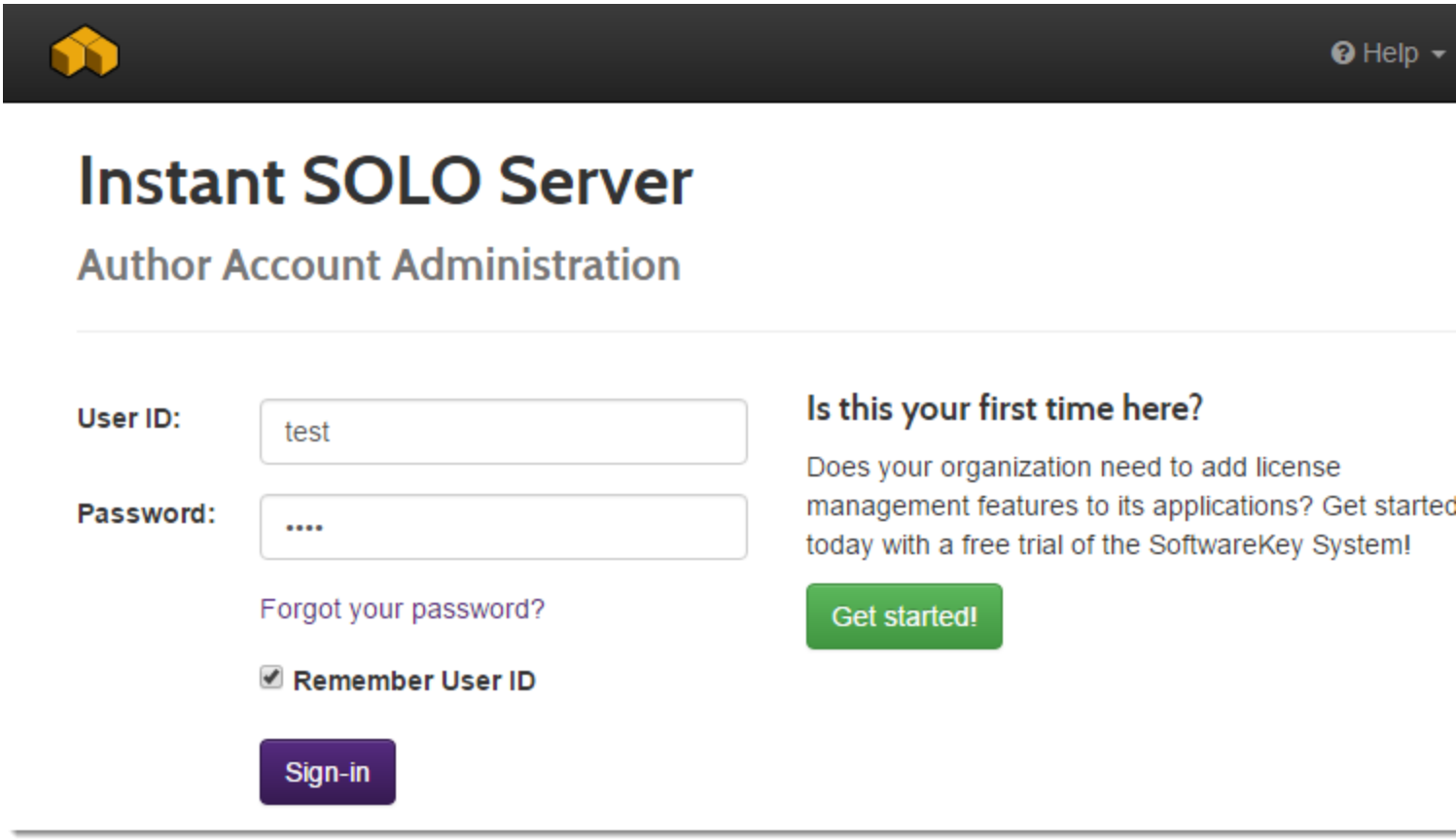
To activate the Sample Licensing Application automatically using SOLO Server, click on Activate Online. A new dialog will prompt for a License ID and Password (both are required). Specifying an installation name is optional, but helpful when performing multiple installations for different machines and/or users.



For ease of use, all of the Protection PLUS 5 SDK sample applications use a generic Test Author account with its own Encryption Key ID. We will use this Test Author account on Instant SOLO Server to generate the License ID and password necessary to activate the Licensing Sample

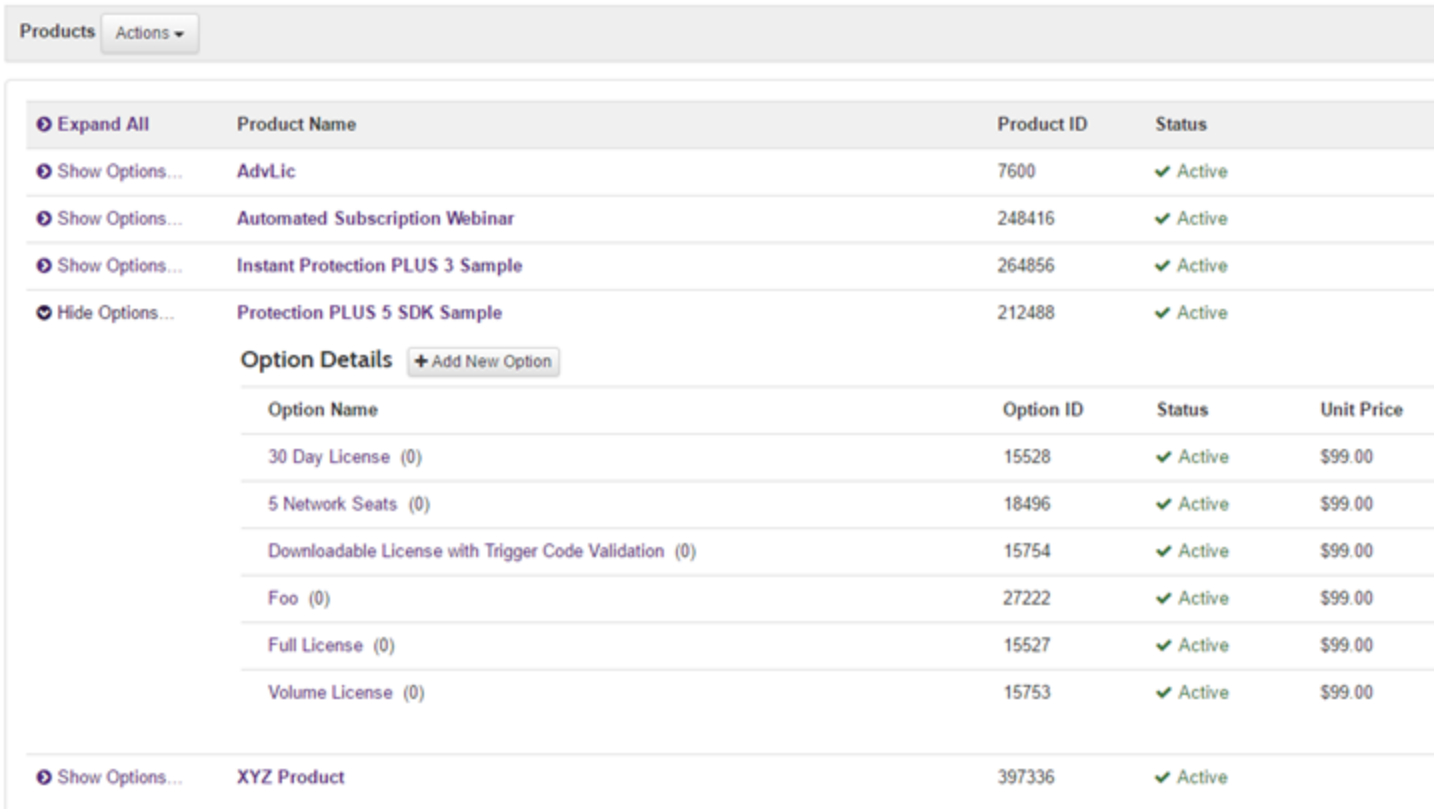
To log intoInstant SOLO Server, visit https://secure.softwarekey.com/solo/authors/Default.aspx. Use the Test Author credentials given below:

- User ID: test
- Password: test

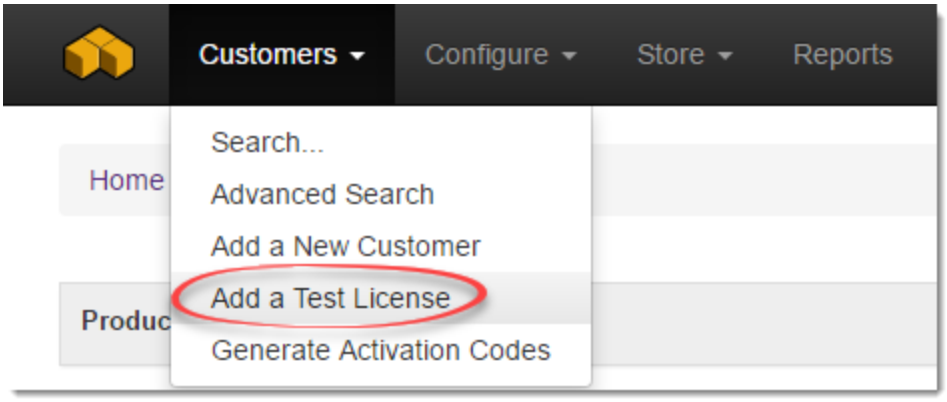


Once logged in, go to the menu *Configure / Products*.

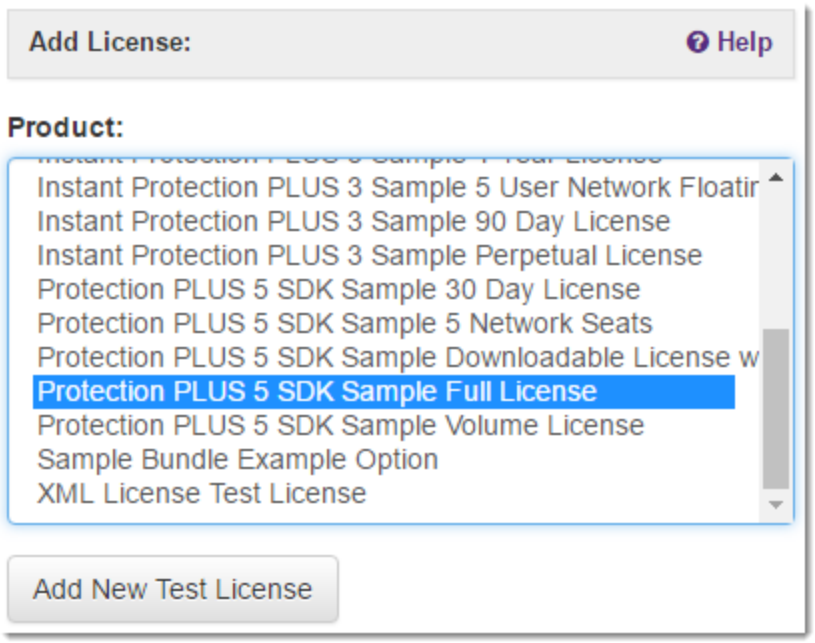
The Product List page will show all of the products available to the Test Author. Expand 'Show Options' beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for activating the Licensing Sample application. For this tutorial, we will generate a license for the Full License.



To add a test license for this product option, mouse over Customers in the navigation bar, and click 'Add Test License'.

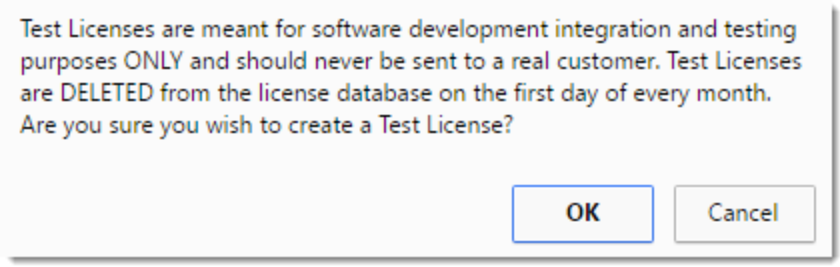


Select the Protection PLUS 5 SDK Sample Full License and click Add New Test License.



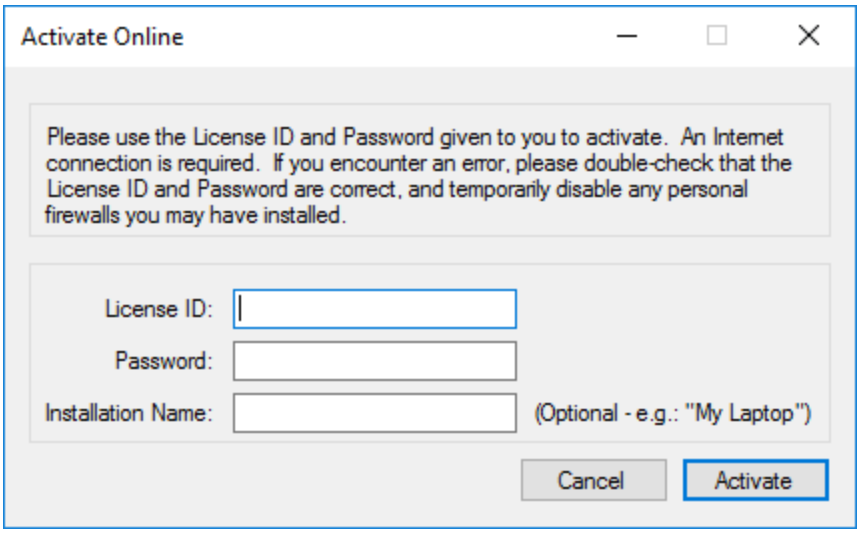
Important:

Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.

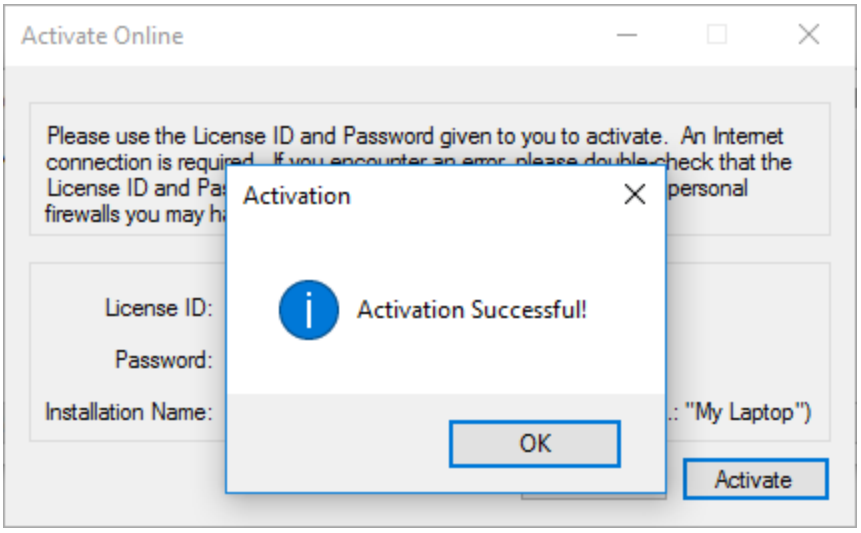


The Test License has now been created. We now have a License ID and Password to continue testing the Licensing Sample.

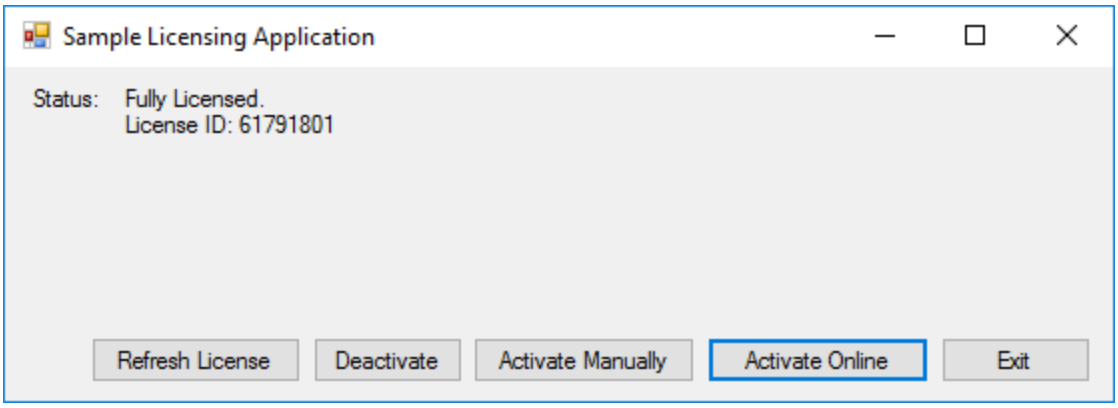
Return to the activation prompt for our Licensing Sample. Enter the License ID and Password in their respective fields and, optionally, an Installation Name. Click Activate to communicate with Instant SOLO Server for license validation.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.

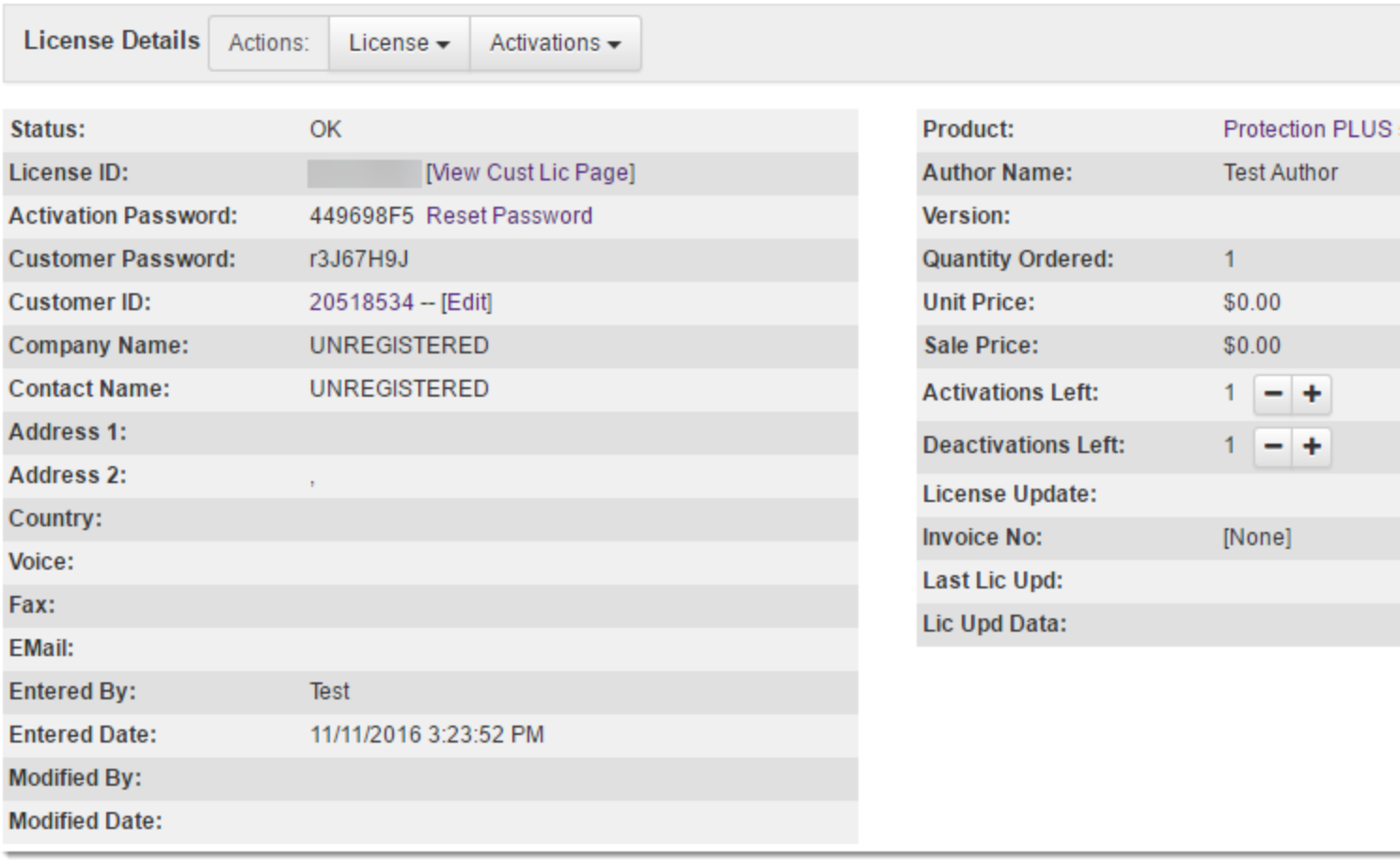


The License Status will now show as Fully Licensed.

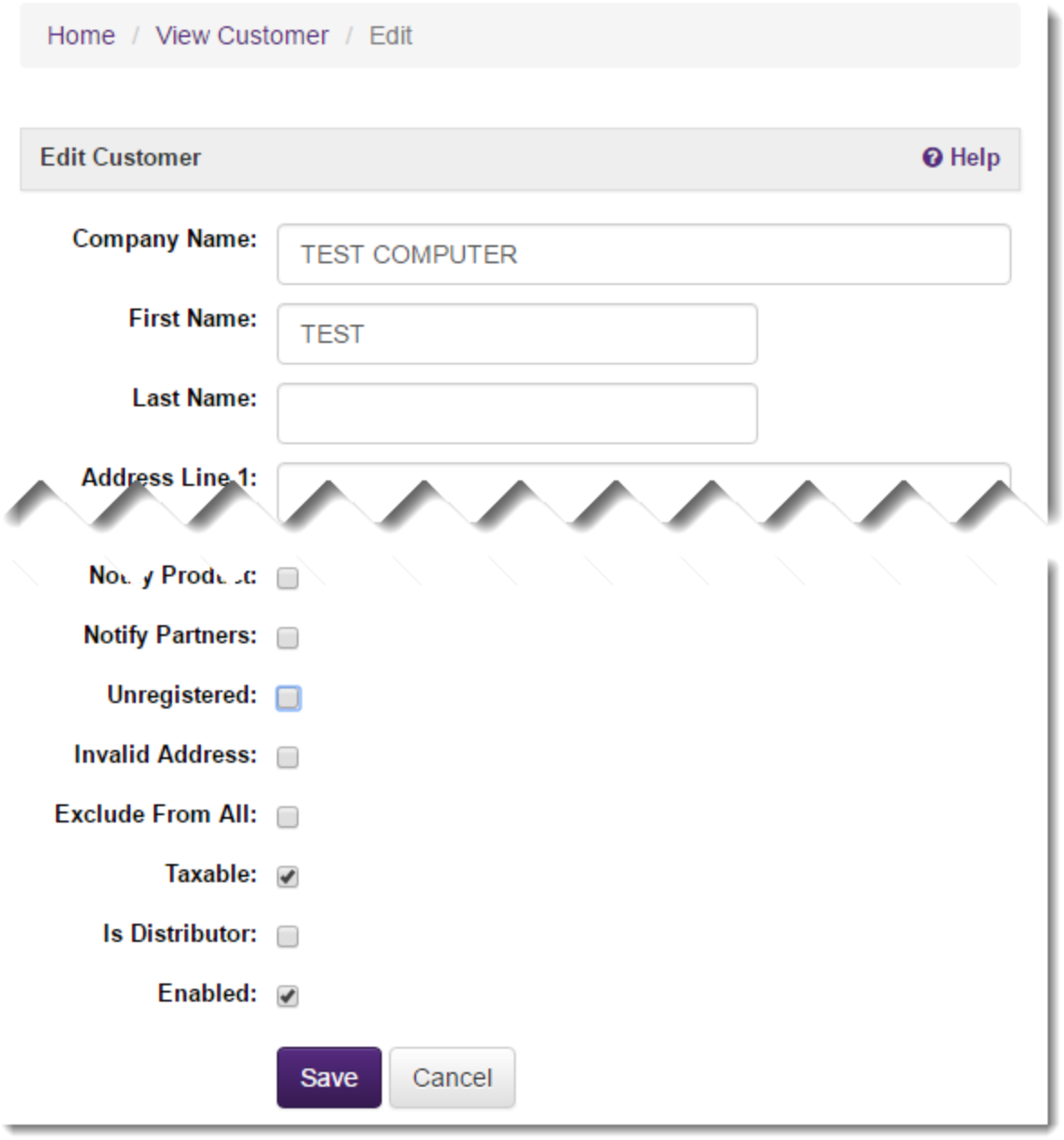


Step 3- Refresh a License

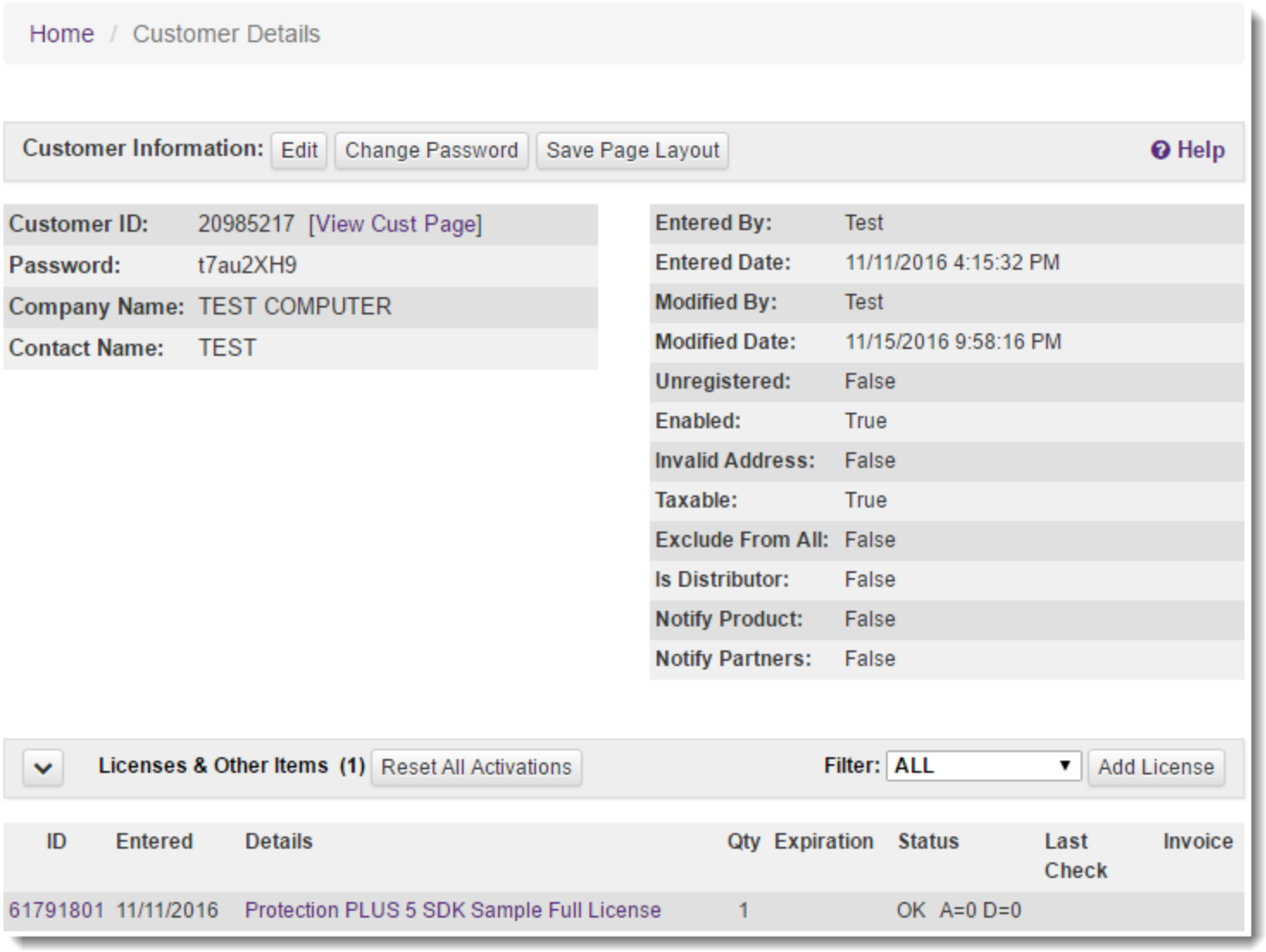
A license can be refreshed with updated information sent from Instant SOLO Server. We can modify customer details from the License Details page on Instant SOLO Server. On the License Details page, click Edit, located to the right of the Customer ID.



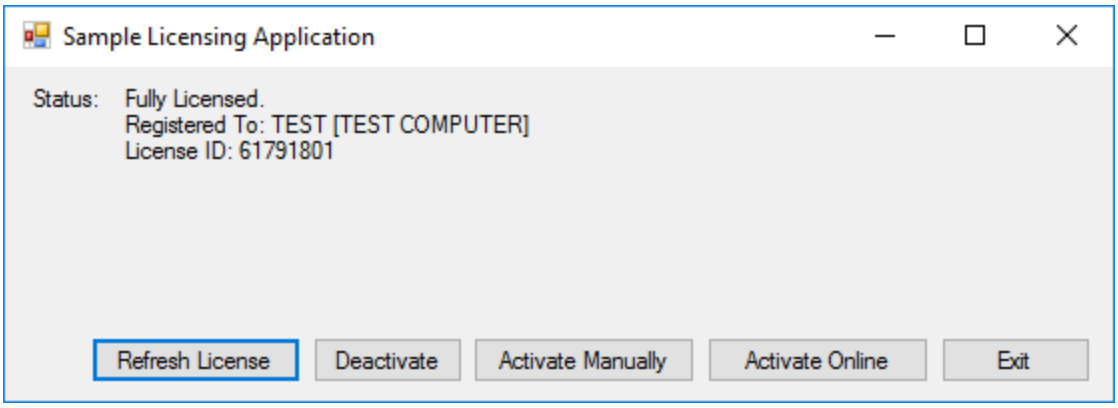
Modify the Company Name and additional fields as you please. Remove uncheck the check box setting to the right of Unregistered. Scroll to the bottom and click Save.



After the Customer Details have been saved, the page returns to the Customer Information page. This page includes customer details we have edited in addition to licenses and previous orders.

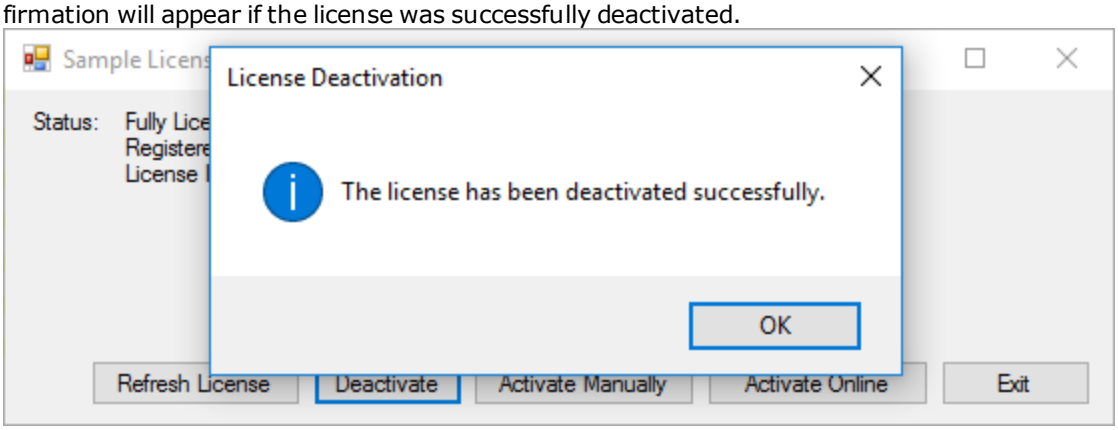


Return to the Licensing Sample Main Dialog to proceed with a license refresh. Click Refresh License. If the licensed successfully refreshed, the License Status will now update with the modified customer details.

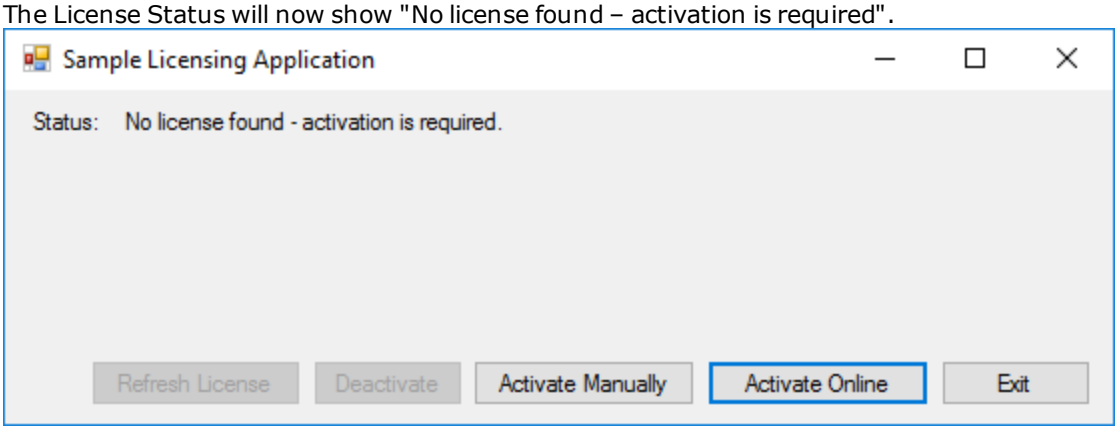


Step 4 - Process a Manual Activation

If you previously activated the Licensing Sample online using Instant SOLO Server, you will need to deactivate the license before testing manual activations. In the Main Dialog of the Licensing Sample, click Deactivate. A confirmation will appear if the license was successfully deactivated.

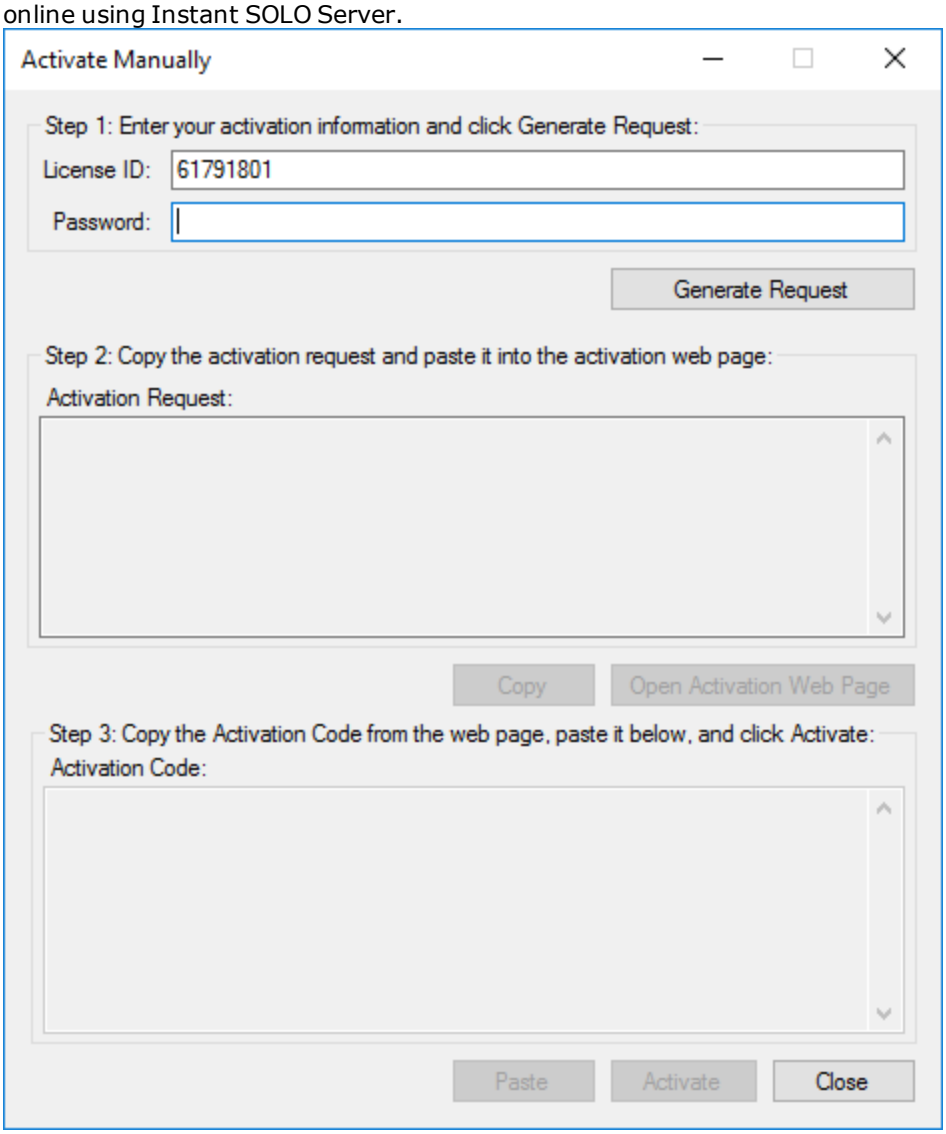


The License Status will now show "No license found – activation is required".

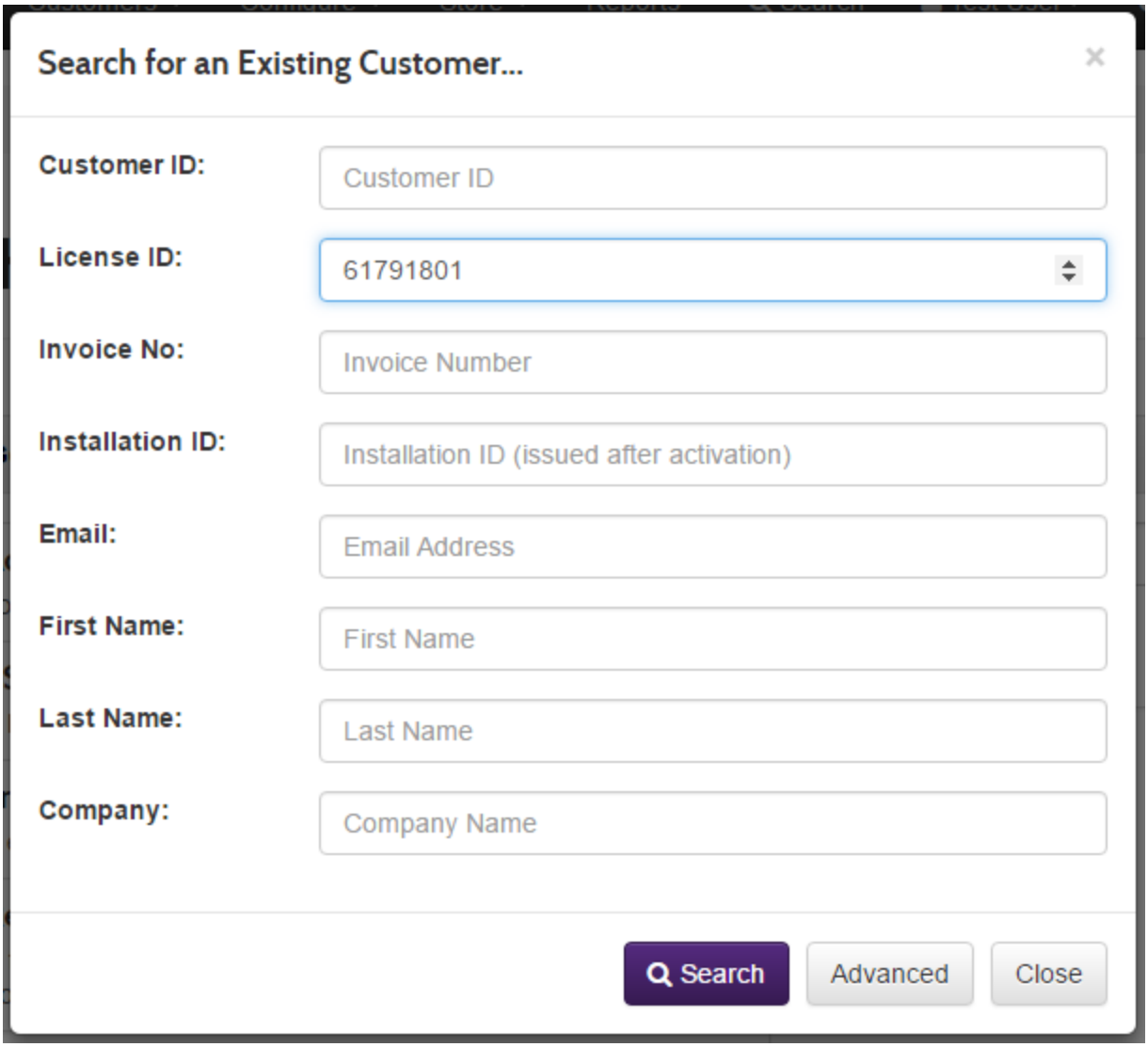


Click Activate Manually to proceed with testing a manual activation. A manual activation allows for activations from another computer or by e-mail. Follow the instructions in Step # 2 to generate a new license. Alternatively, we can

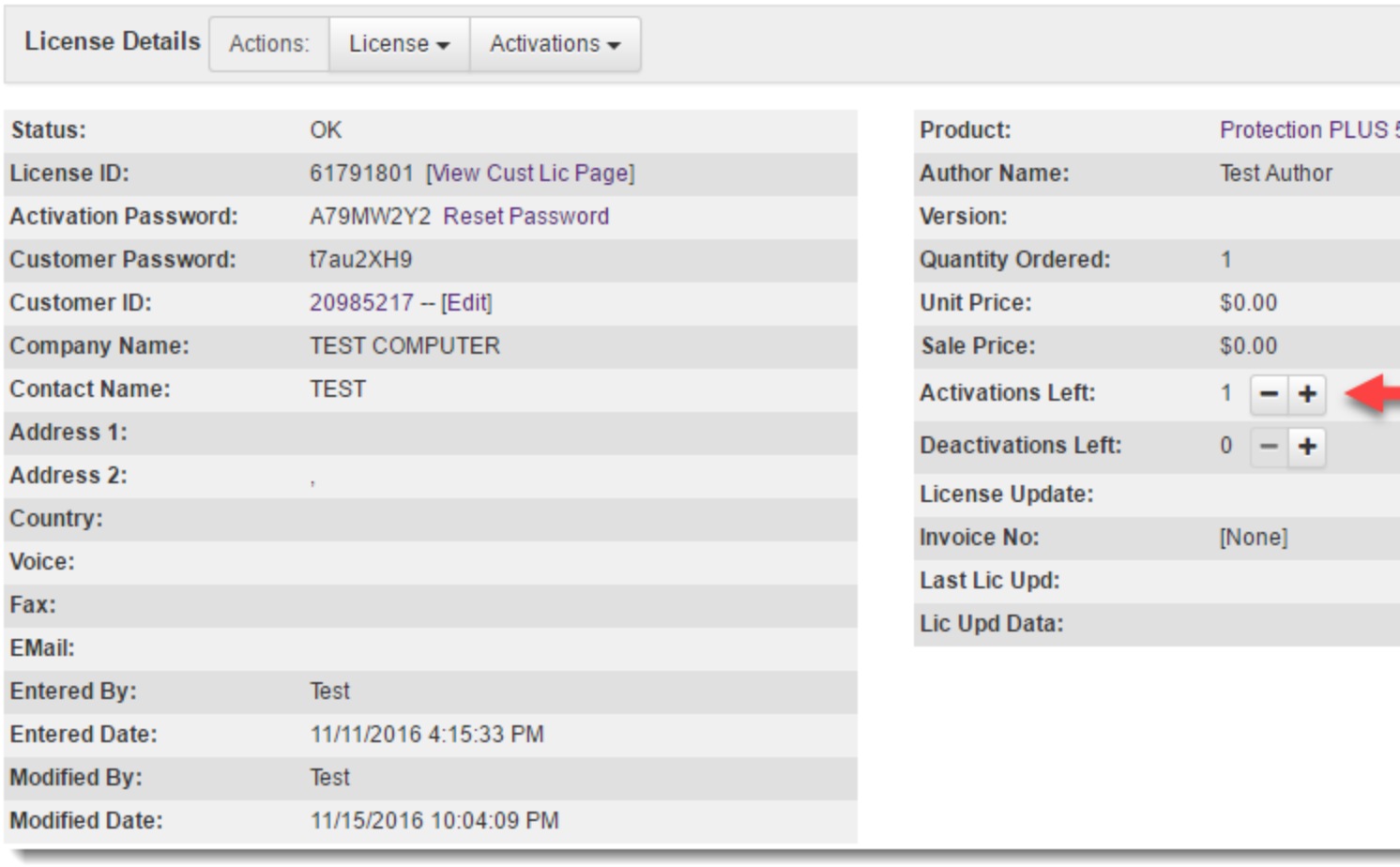
also check the number of activations remaining for the License ID generated when we tested automatic activations online using Instant SOLO Server.



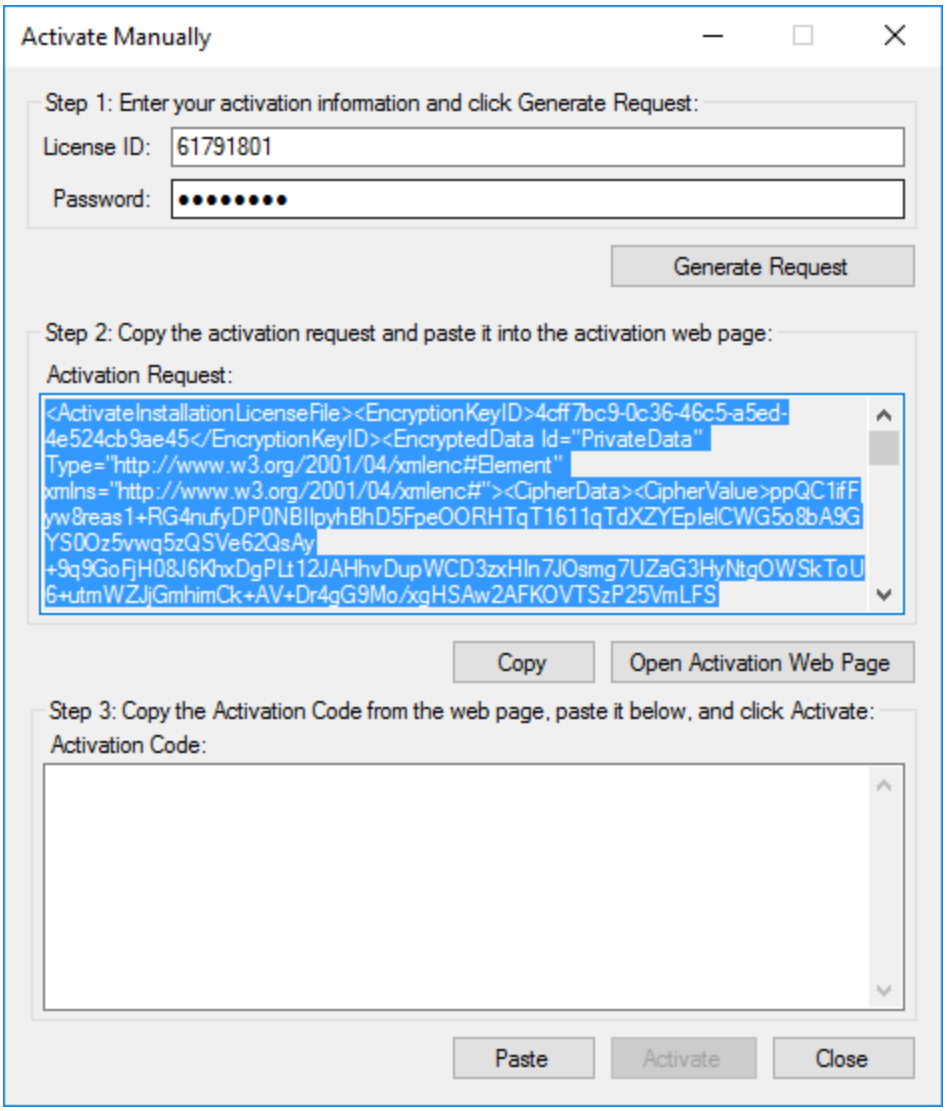
Return to Instant SOLO Server using the Test Author account. Perform a quick search for the License ID previously used to activate online (If you no longer have this number, proceed to create a new License ID as shown in Step # 2).



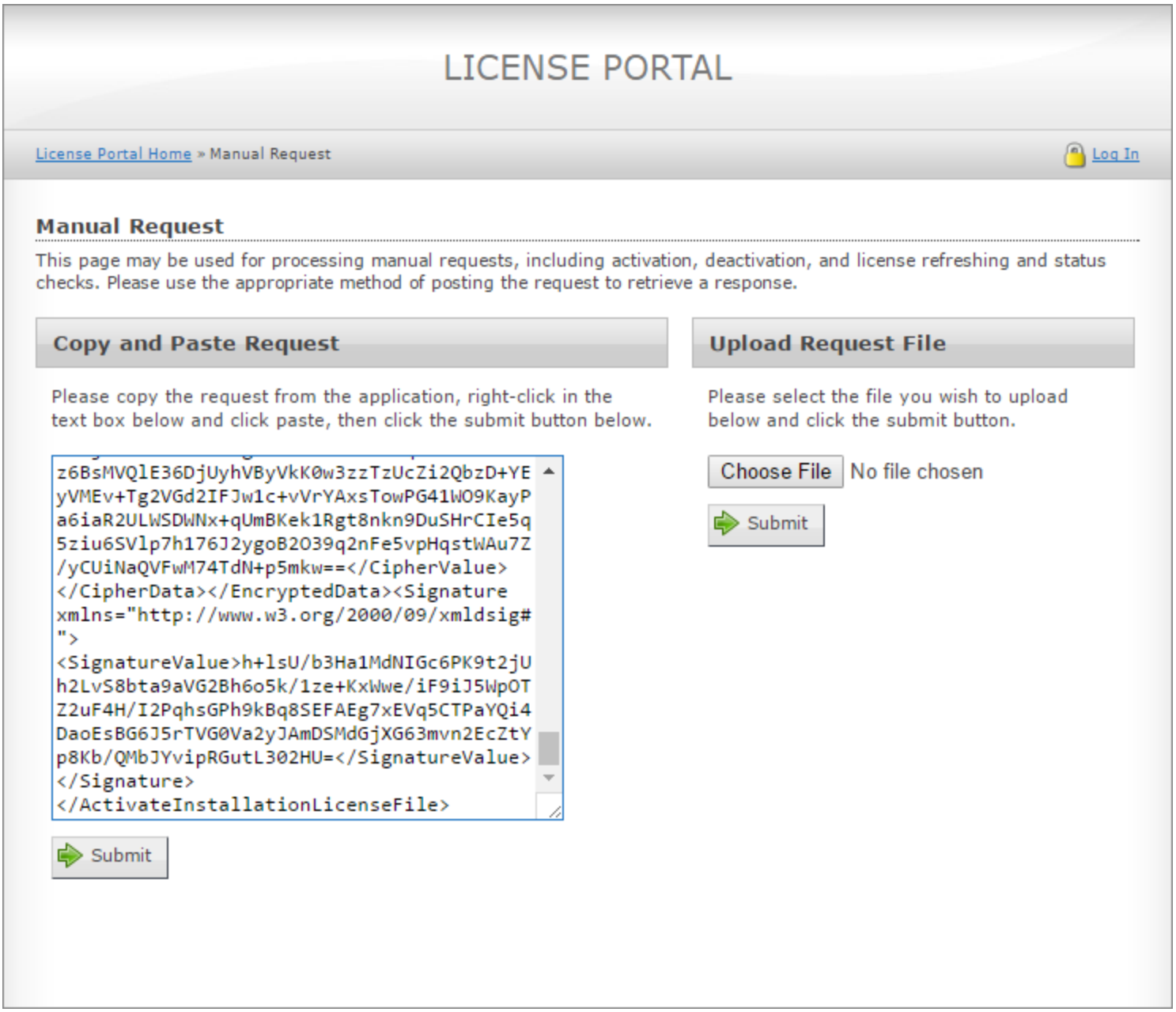
When we clicked on deactivate license, Instant SOLO Server automatically incremented the amount of activations left by 1. This setting was predefined in the Product Option ID we selected for testing. If you need to manually increase the number of activations left, you can increment this number by clicking the + button to the right of the Activations Left field.



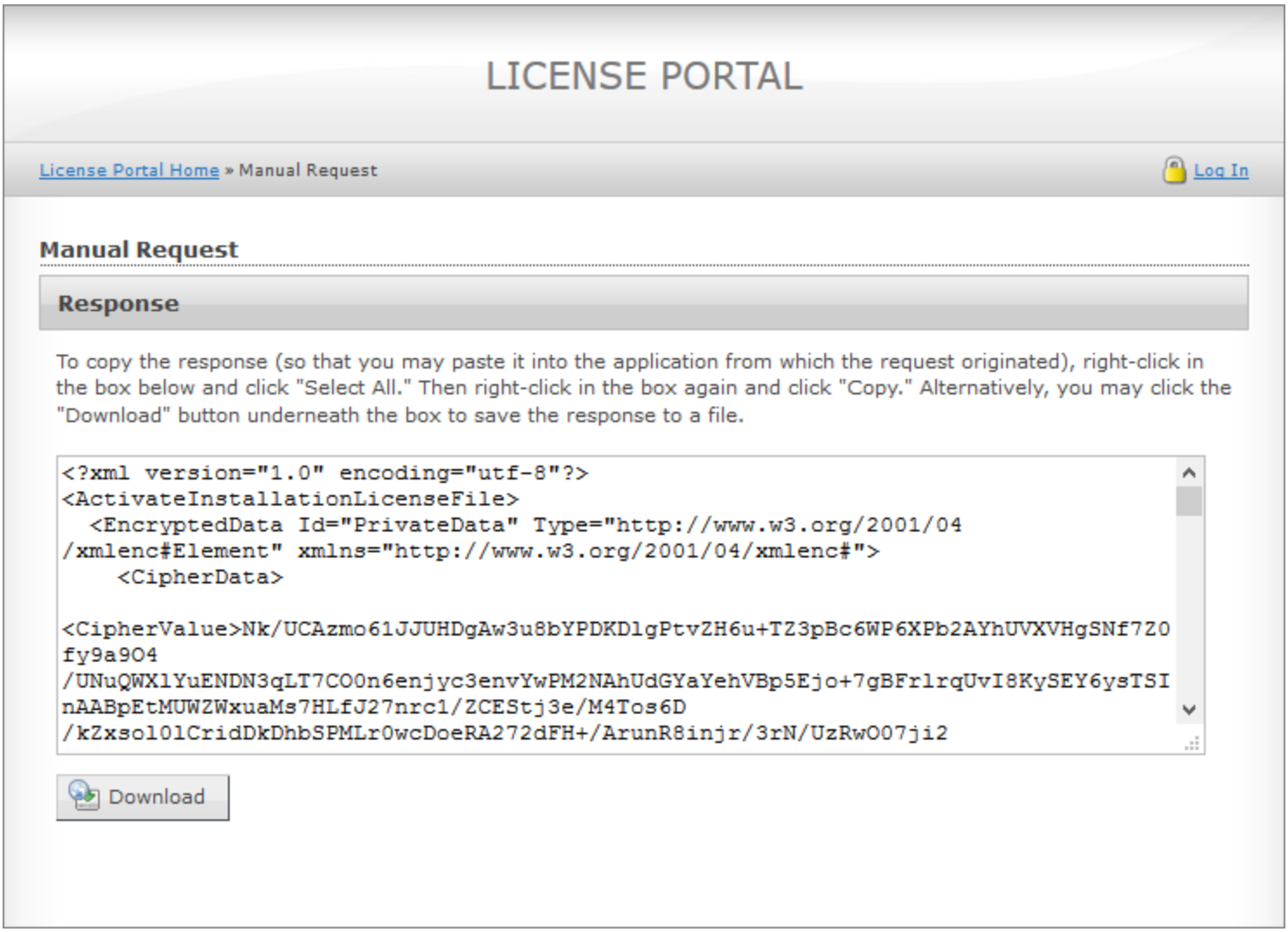
Return to the Activate Manually for our Licensing Sample. Enter the License ID and Password in their respective fields and click Generate Request.



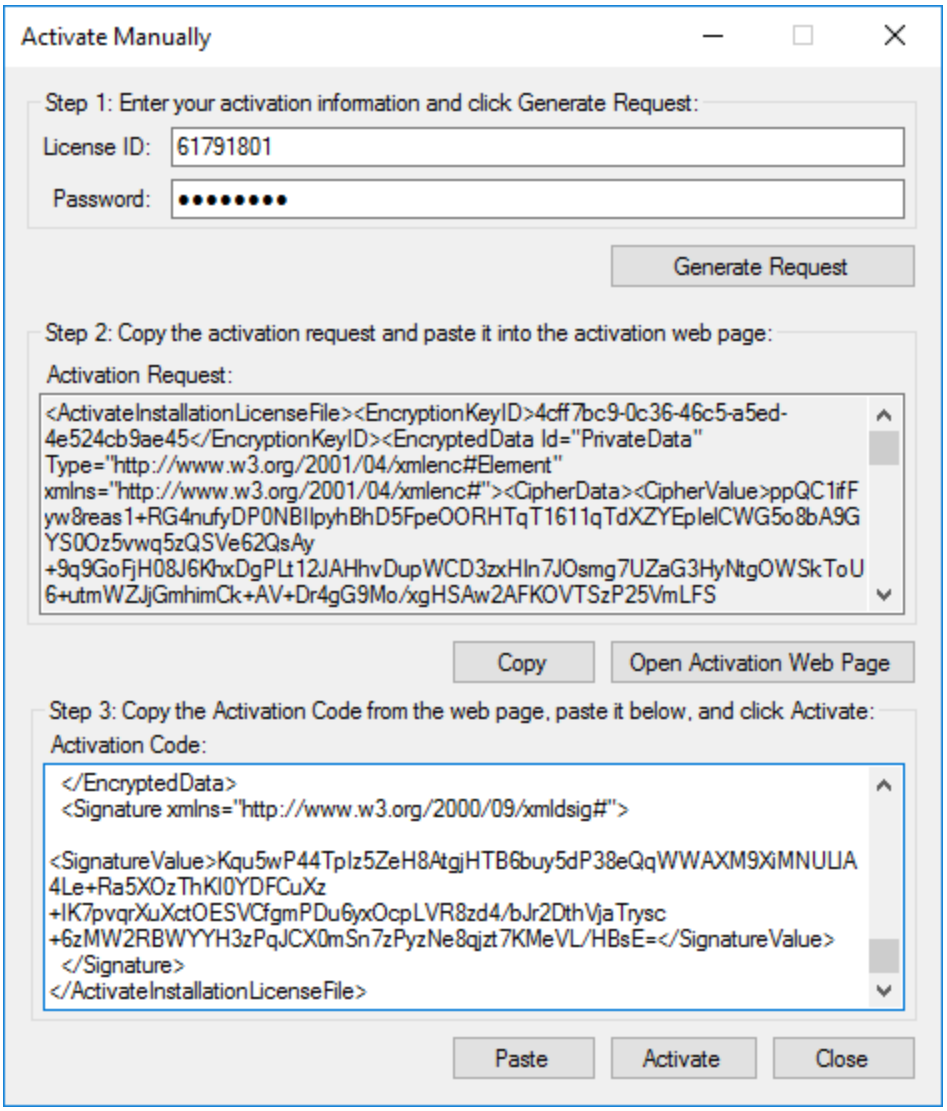
This dialog will generate an encrypted activation request to be processed by the activation server. Click Copy to copy this data to the clipboard. Next, click Open Activation Web Page. This web page is accessible from another computer with access to the Internet in the event that an offline workstation needed to be activated. Once you arrive to the License Portal page, paste the encrypted activation request data, and click Submit.



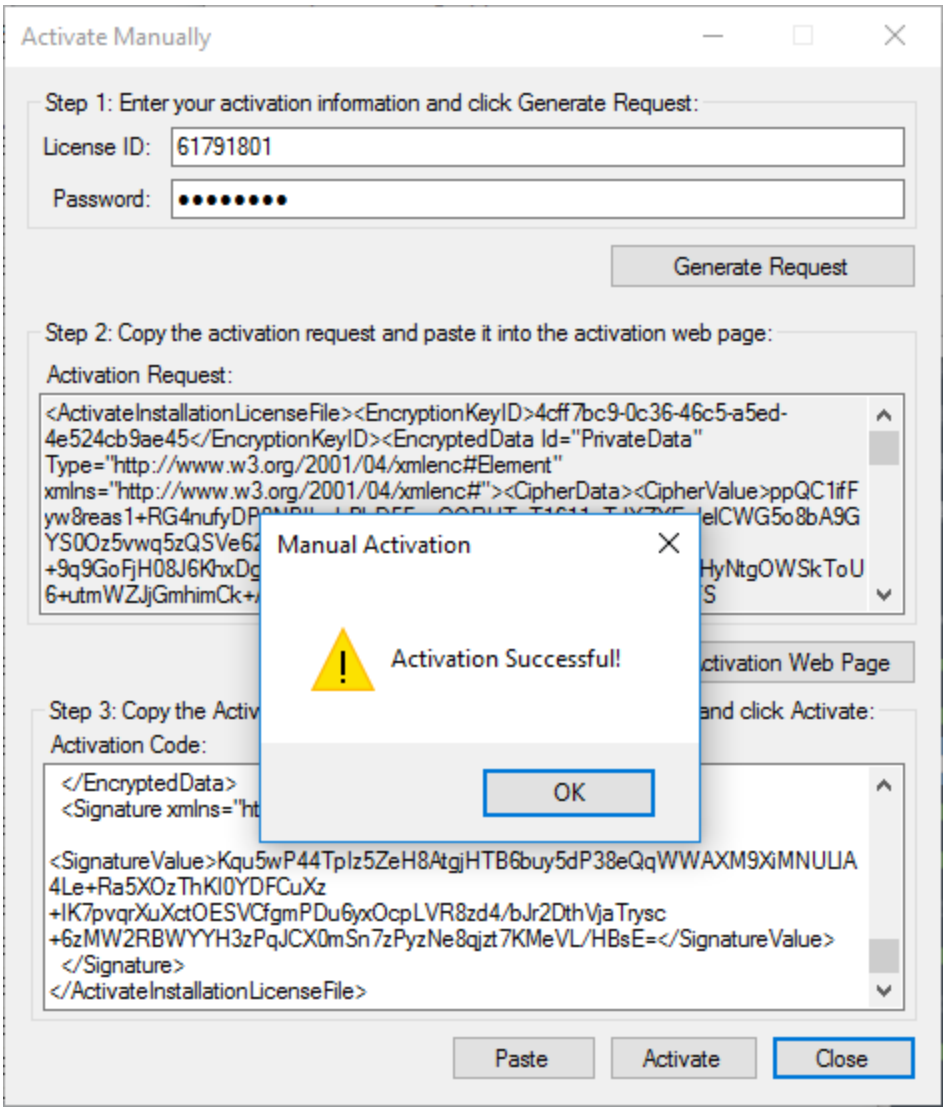
The License Portal will now generate an encrypted response. Copy this data to later paste in the Activation Code field of the activation window.



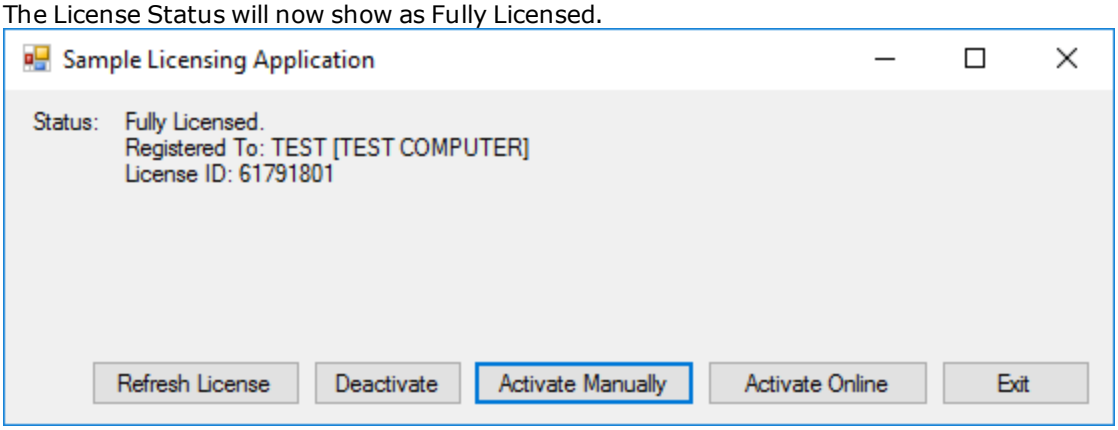
Return to the activation window and paste the encrypted response into the Activation Code field and click Activate.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



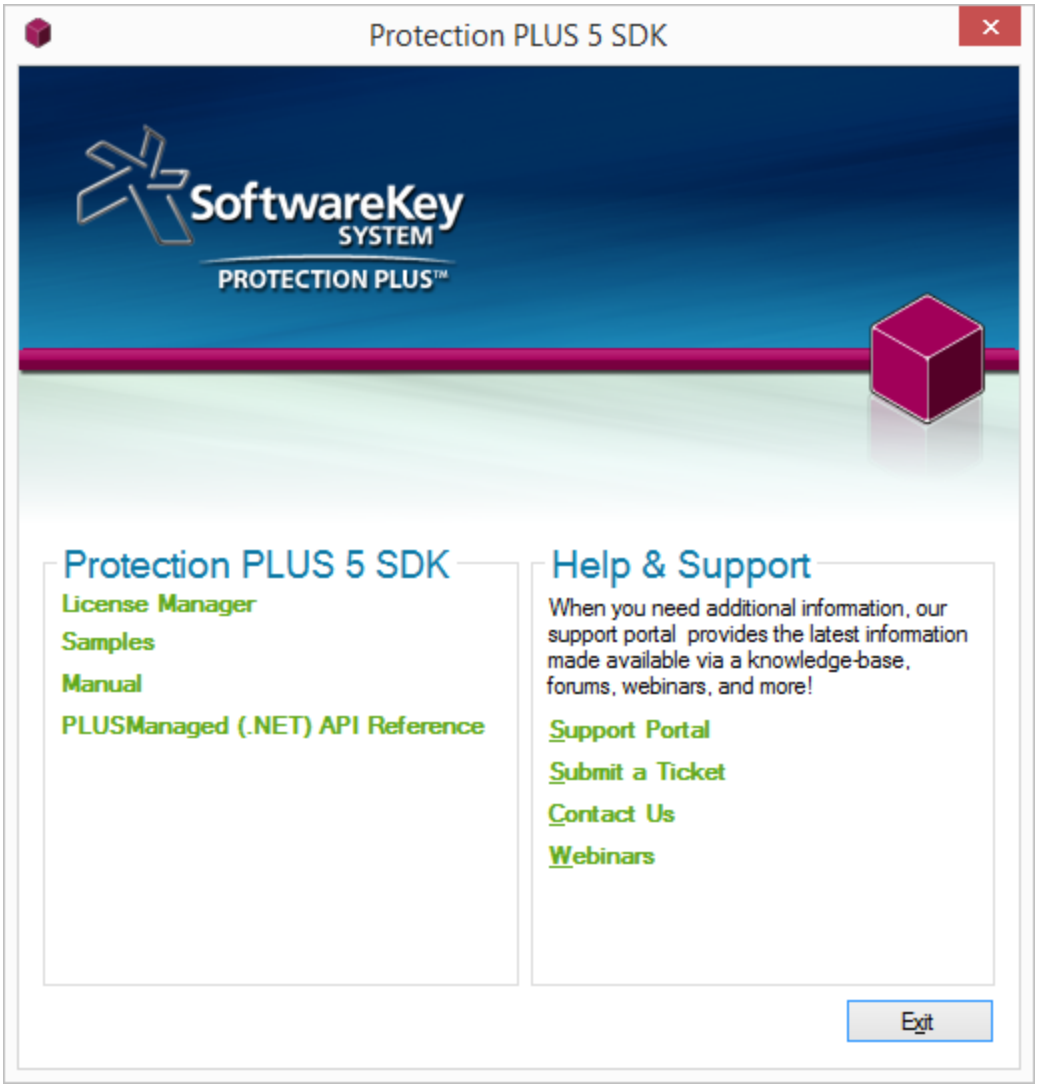
The License Status will now show as Fully Licensed.



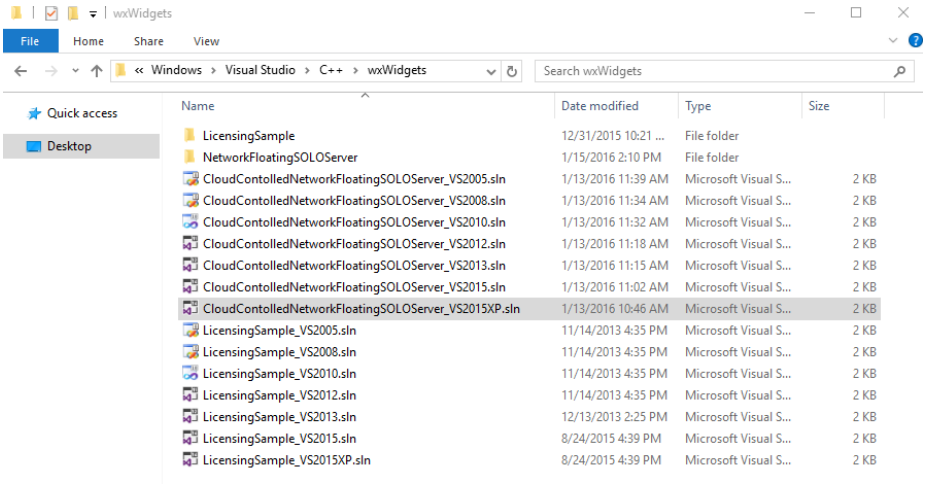
PLUSNative Cloud-Controlled Network Floating Licensing with SOLO Server

Step 1 – Opening the Sample Project

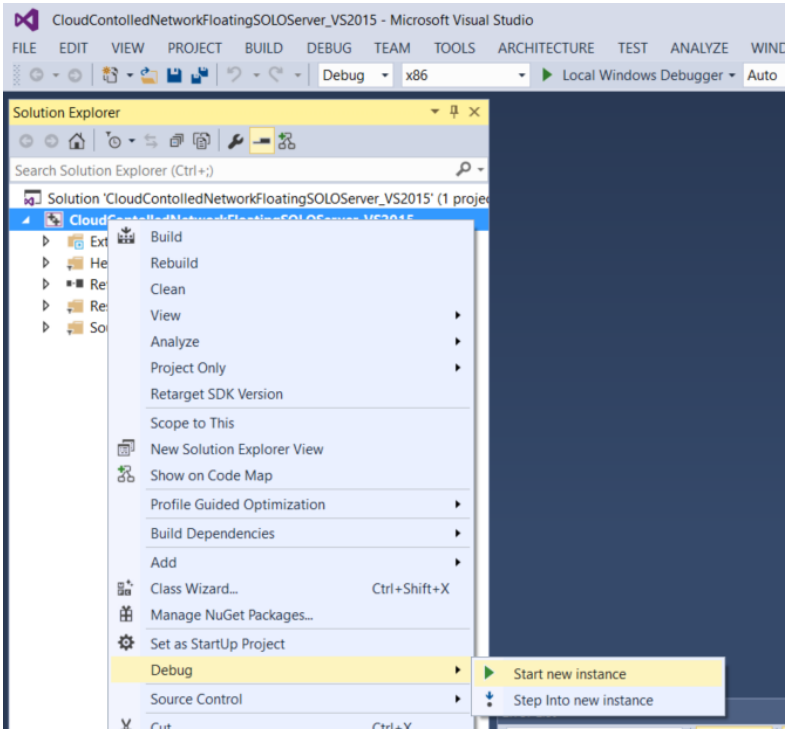
Open the PLUSNative CloudControlledNetworkFloatingSOLOServer*.sln file corresponding to your version of Visual Studio. The Samples directory can be opened by running Protection PLUS 5 SDK from your start menu/screen, and clicking on the Protection PLUS 5 SDK Samples link.



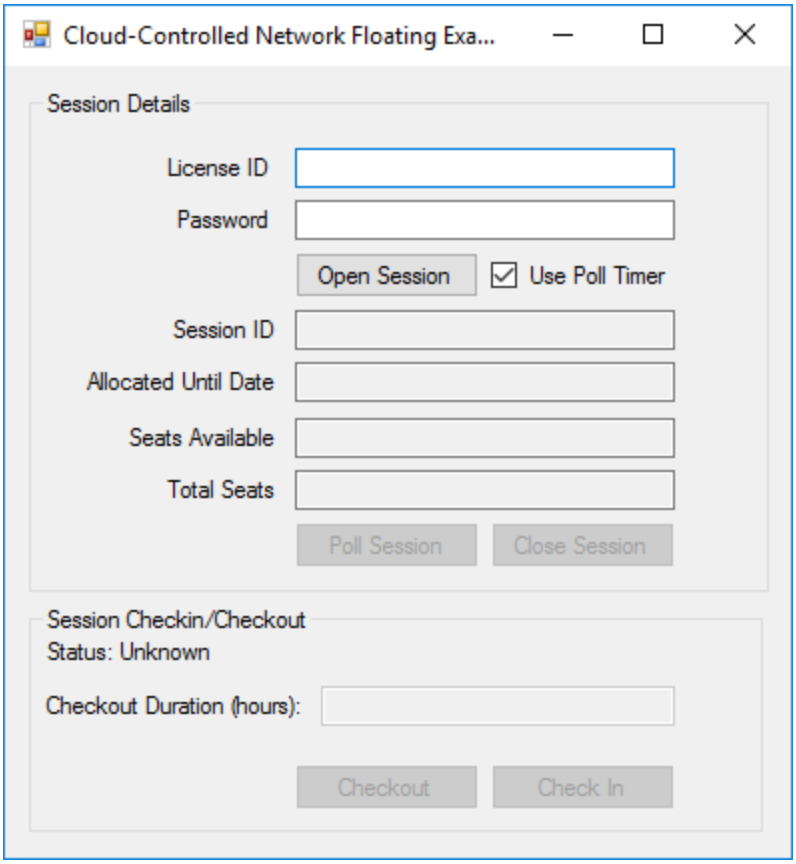
For this example we will be using the wxWidgets (C++) Sample in Visual Studio.



Expand the Solution folder and locate the CloudContolledNetworkFloatingSOLOServer project. Right click this project and select Debug > Start New Instance



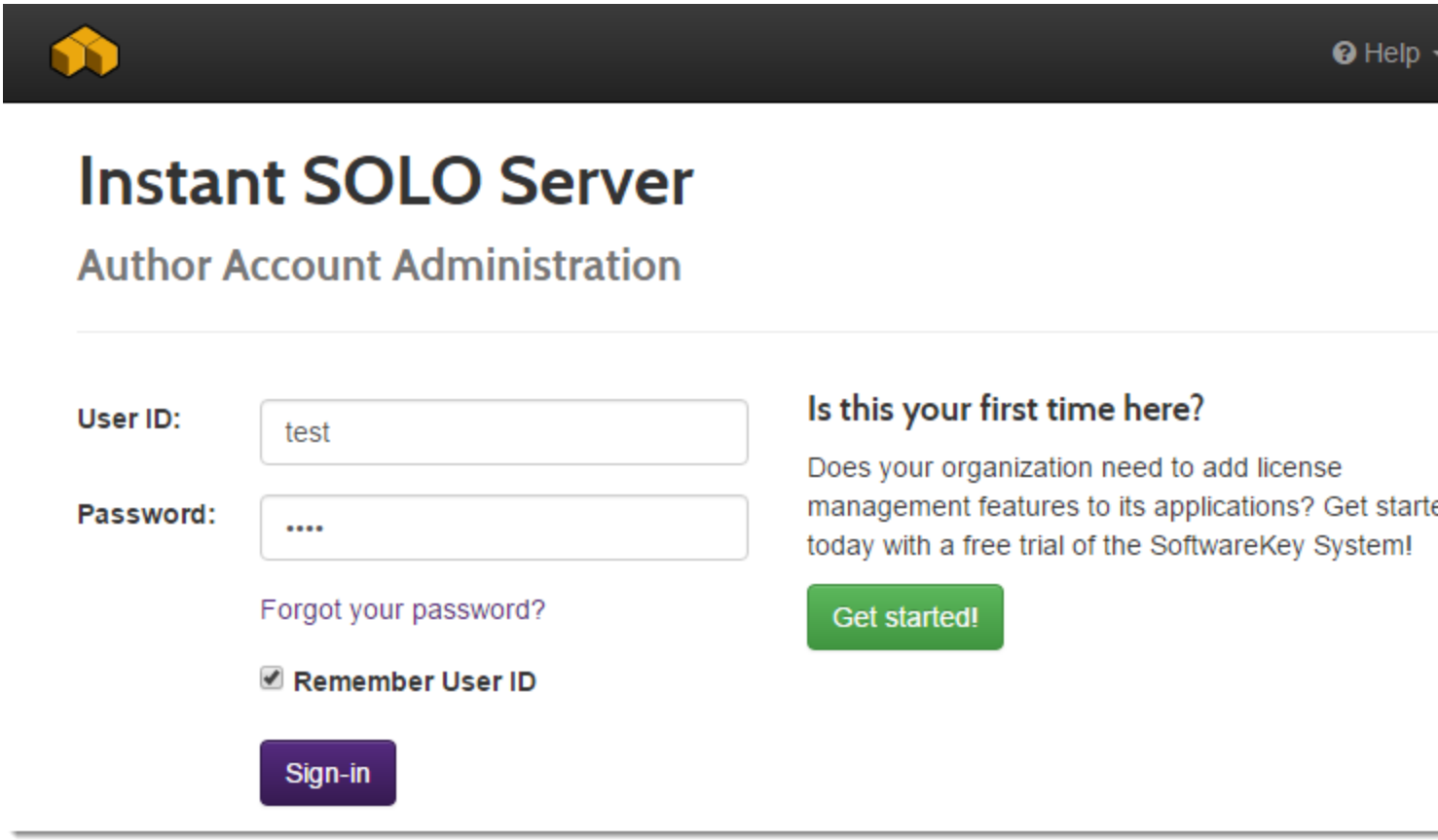
The sample application will execute.



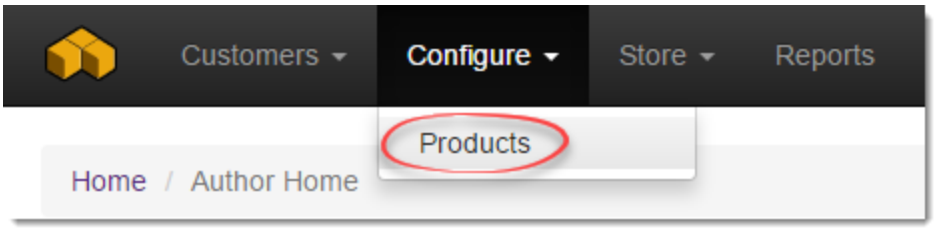
Step 2 – Opening a session with Instant SOLO Server

For ease of use, all of the Protection PLUS 5 SDK sample applications use a generic Test Author account with its own *Encryption Key ID* . We will use this Test Author account on Instant SOLO Server to generate the License ID and activation password necessary to activate the Sample. To log into Instant SOLO Server, visit **https://secure.softwarekey.com/solo/authors/Default.aspx**. Use the Test Author credentials given below:

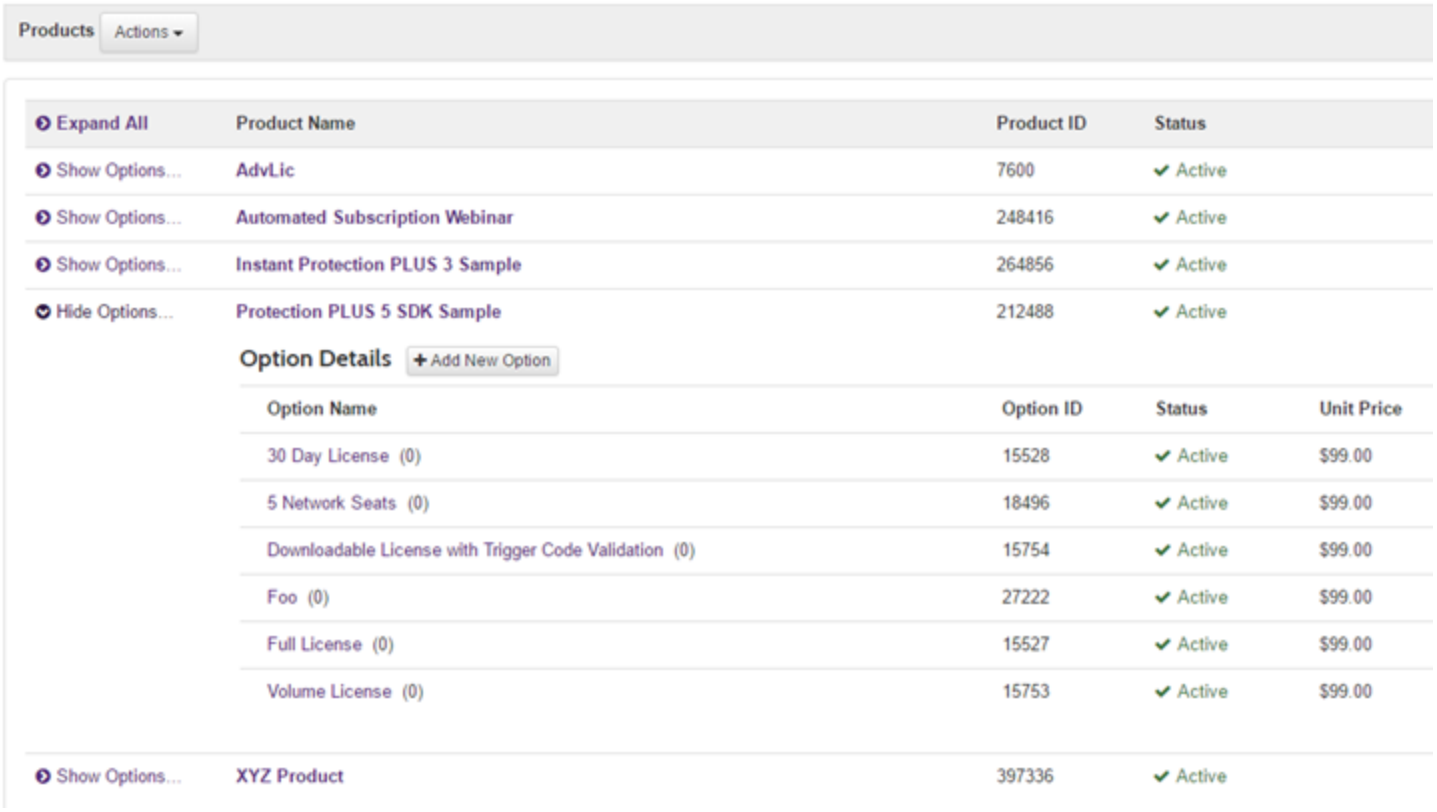
Login ID: test
Password: test



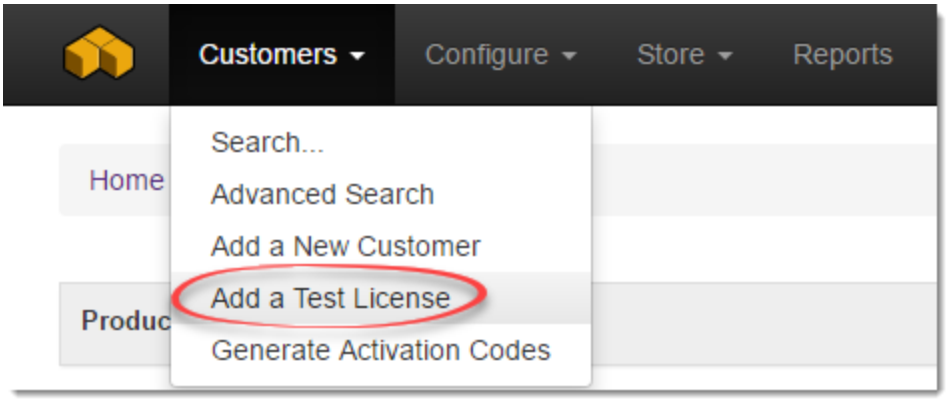
Once logged in, go to the menu *Configure / Products*.



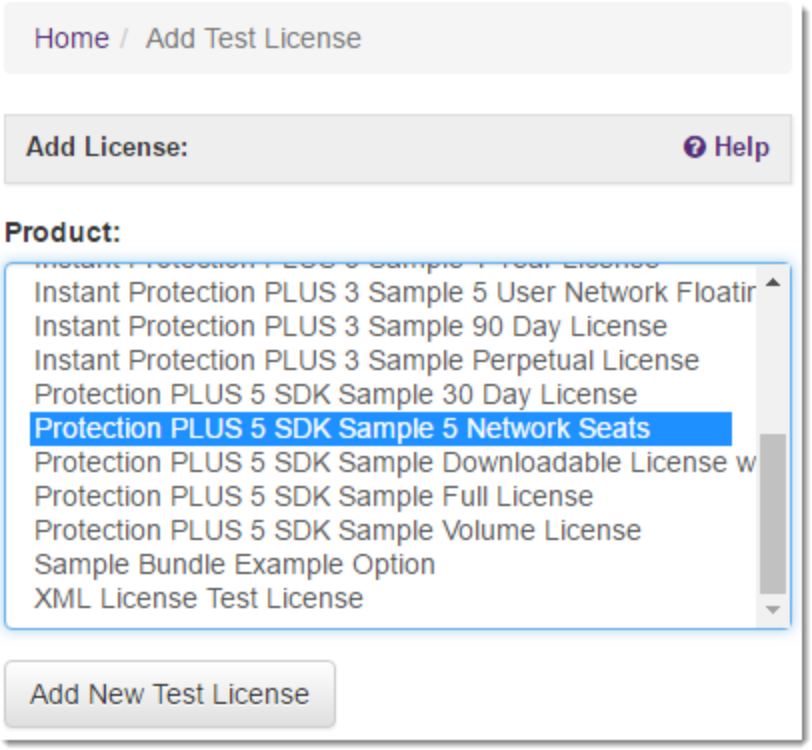
The Product List page will show all of the products available to the Test Author. Expand Show Options beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for activating the sample application. For this tutorial, we will generate a license for the 5 Network Seats.



To add a test license for this product option, mouse over Customers in the navigation bar, and click "Add Test License".



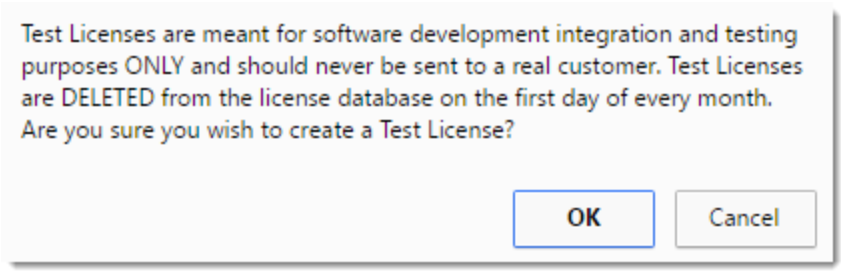
Select the Protection PLUS 5 SDK Sample 5 Network Seats and click *Add New Test License*.



Important:

Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.

Press OK to continue when prompted.



>

The Test License has now been created. We now have a License ID and an Activation Password to continue testing the sample application.

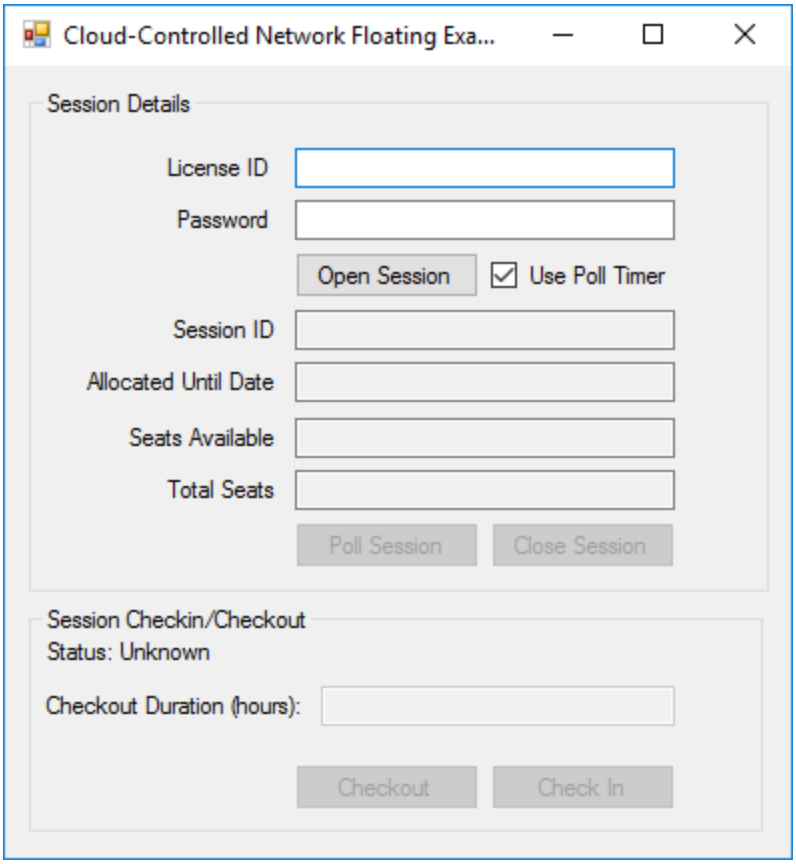
IMPORTANT: This is a test license, which will be deleted from the database on 12/1/2016!

License Details | Actions: | License ▾ | Activations ▾

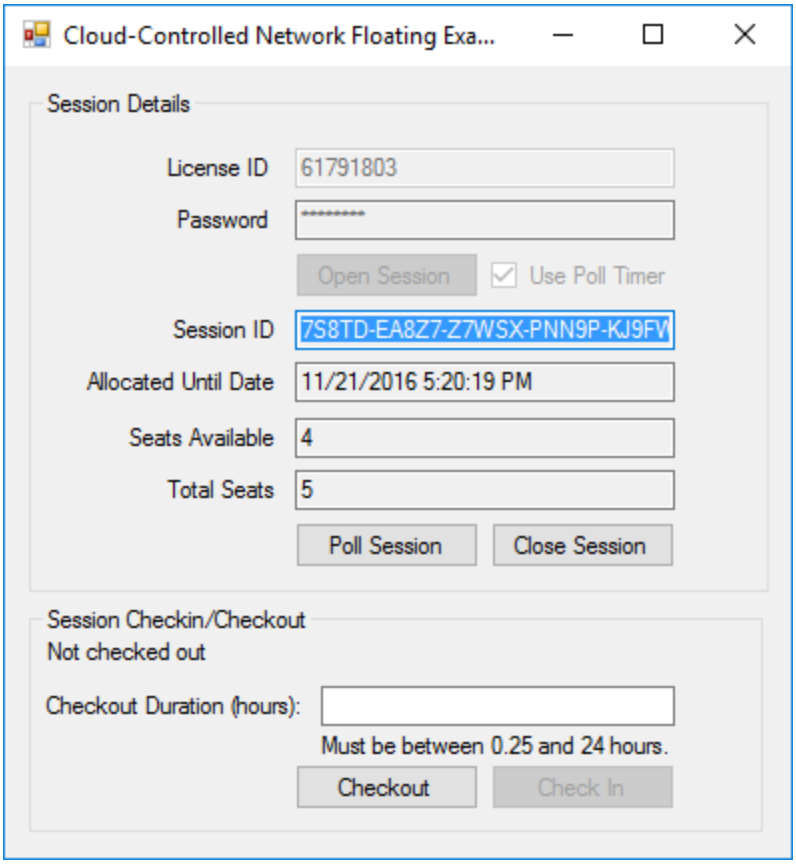
Status:	OK
License ID:	61791803 [View Cust Lic Page]
Activation Password:	992TG459 Reset Password
Customer Password:	872NkFhP
Customer ID:	20985230 -- [Edit]
Company Name:	UNREGISTERED
Contact Name:	UNREGISTERED
Address 1:	
Address 2:	,
Country:	
Voice:	
Fax:	
EMail:	
Entered By:	tochel
Entered Date:	11/21/2016 5:12:30 PM
Modified By:	
Modified Date:	

Product:	Protection PLUS
Author Name:	Test Author Swi
Version:	
Quantity Ordered:	1
Unit Price:	$0.00
Sale Price:	$0.00
Activations Left:	3 − +
Deactivations Left:	Infinite
Allowed Network Seats:	5 (0 currently in
License Update:	
Invoice No:	[None]
Last Lic Upd:	
Lic Upd Data:	

Return to the Cloud-Controlled Network Floating Licensing Sample application. Enter the License ID and Activation Password in their respective fields. Click Open Session to communicate with Instant SOLO Server for license validation and to open a session.



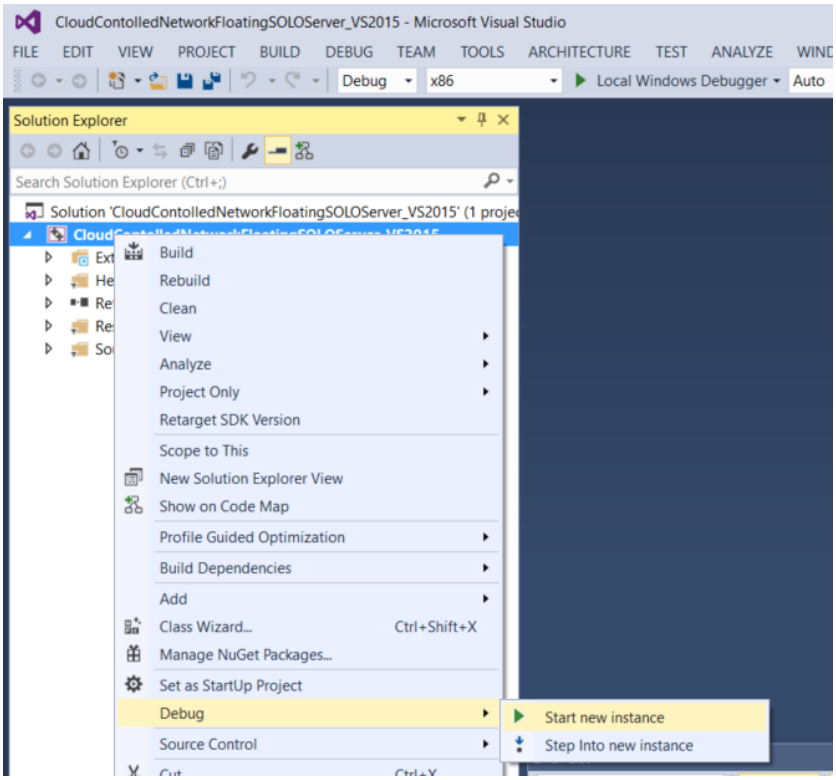
Upon successful activation, the Session ID, Allocated Until Date, Seats Available, and Total Seats edit boxes will be propagated with values received from Instant SOLO Server. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



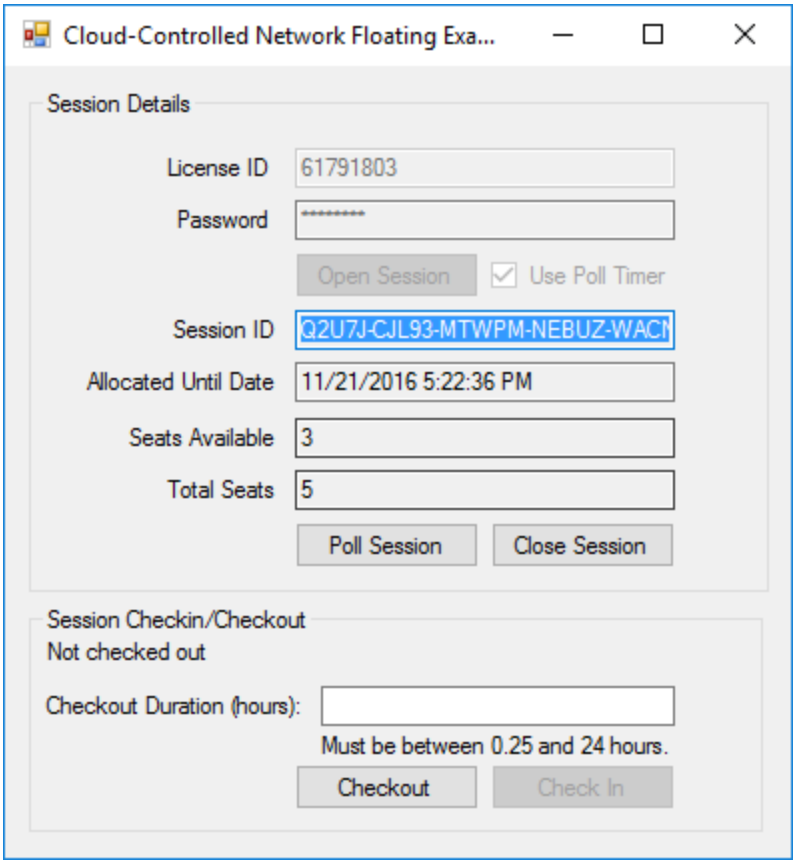
Note the number of Seats Available is 4 out of 5 since we have just used one of the seats with this session.

Step 3 – Opening a Second Session

Open another instance of the sample application, but do not close the instance that is currently open. In Visual Studio right click this sample and select Debug > Start New Instance



To open a session for the second instance of the sample application, enter the same License ID and Activation Password used in the previous instance into their respective fields in this second instance. Click Open Session to communicate with Instant SOLO Server for license validation and to open a session.



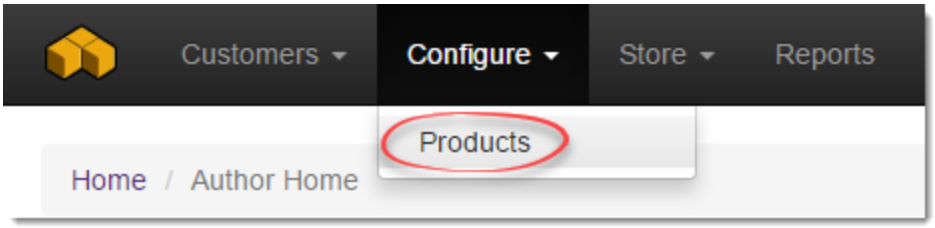
Note there are now 3 seats available of 5 total seats.

Step 4: Finding the Settings in SOLO Server

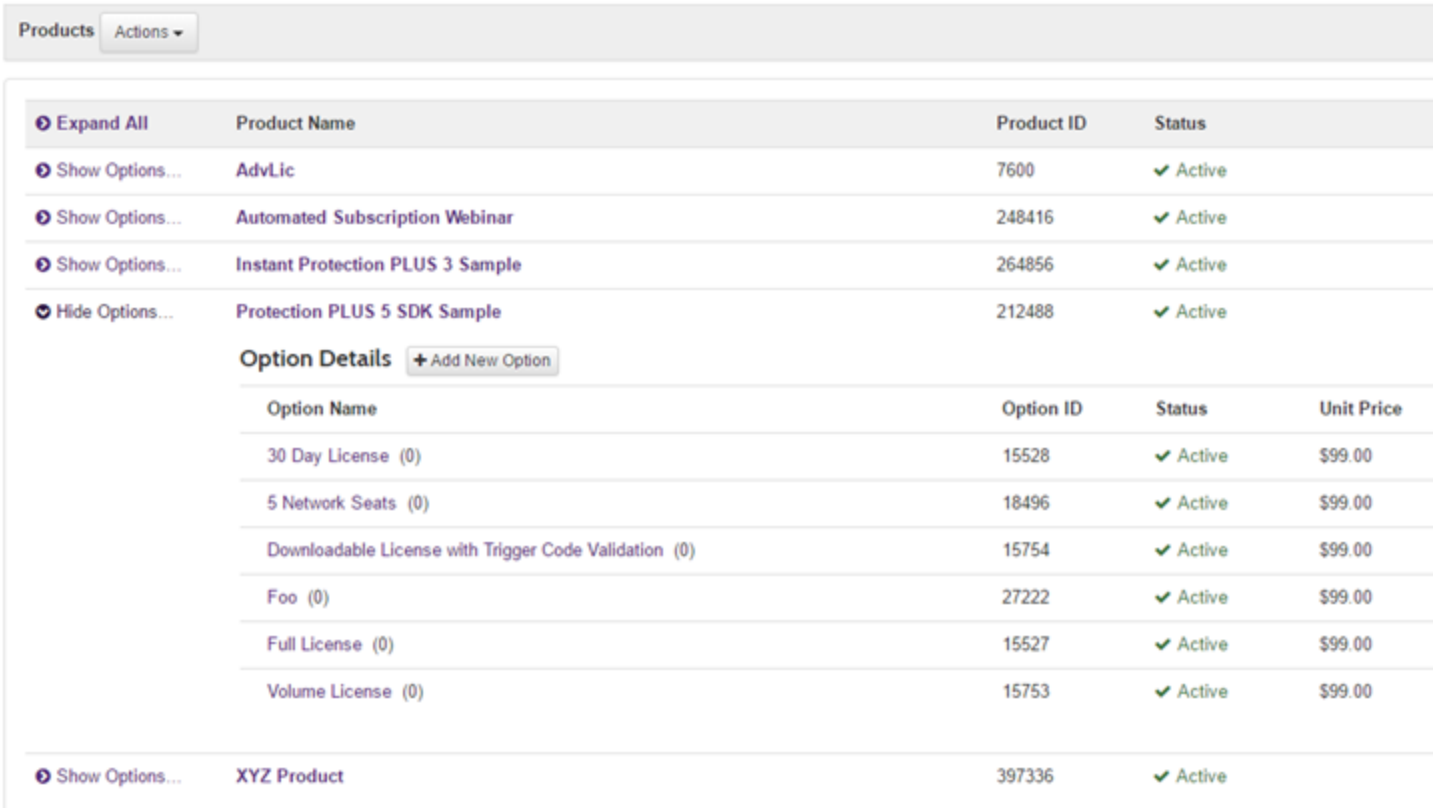
Important:

These settings cannot be accessed with the Test Author account, you will be able to access this **ONLY** when your account is fully activated and enabled for Cloud-Controlled Network Floating Licensing.

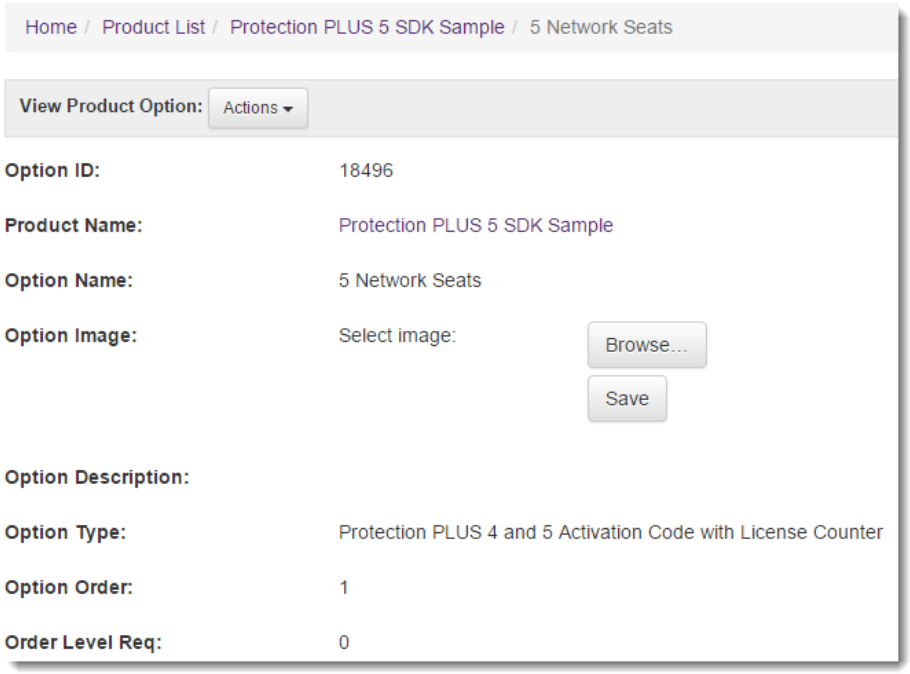
Returning to SOLO Server again, go to the menu *Configure / Products*.



The Product List page will show all of the products available to the Test Author. Expand Show Options beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for the sample applications.



Click on the link for the 5 Network Seats option in Instant SOLO Serverwhich will take you to the details page for this option.



Click the Edit button next to "View Product Options:".

Product Activation and Licensing Details:

Activations per U/M: 3

☑ **Issue License**

☑ **Issue Installation ID**

☑ **Allow Deactivations**

Deactivations per U/M: -1

(-1=Unlimited)

☐ **Allow Reactivations on the Same Computer**

Please review the documentation

IP Activation Mode: No restrictions

IP Mask: Please Select

Additional Activation Days: 0

Minimum Activation Version:

Minimum Activation URL:

Unlock Threshold: -1

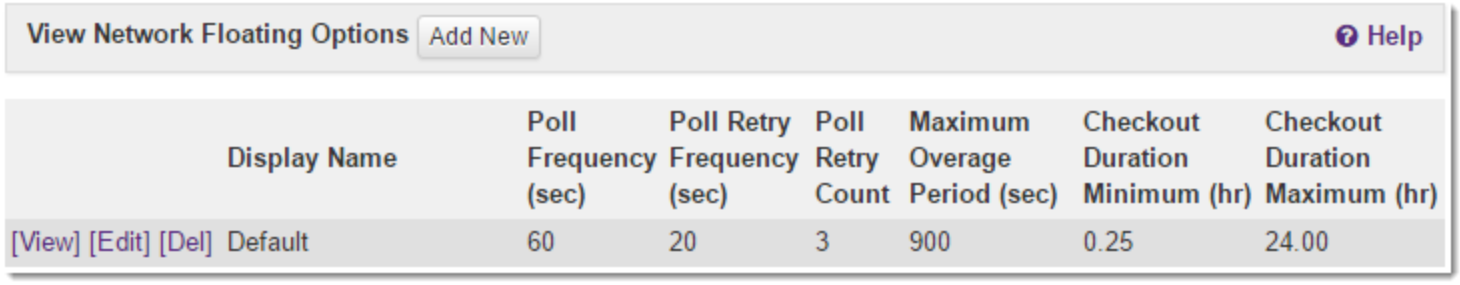
(advanced option)

Trigger Code #: 6

Trigger Code Seed: 400

RegKey 2 Seed: 123

Click on the Configure Network Floating Options link highlighted above.



Here you can view and edit the settings SOLO Server is sending to the application when it opens a session.

LabVIEW Licensing Sample

Tutorial: License a LabVIEW Executable Using Protection PLUS 5 SDK LabVIEW Edition

Visit the SoftwareKey.com Blog for help **Choosing the Best Licensing Method for your LabVIEW Tools**.

Goal

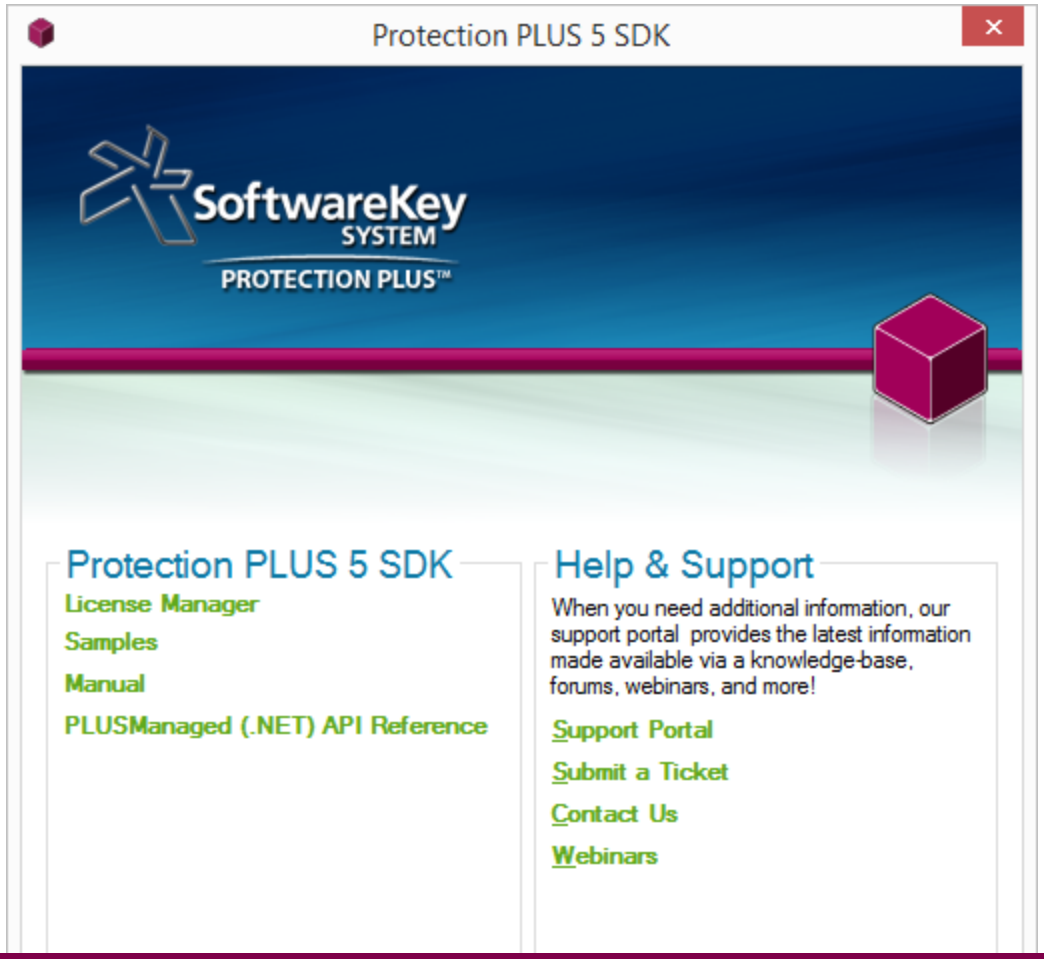
This tutorial will demonstrate how to use the Licensing Sample project using Protection PLUS 5 SDK and LabVIEW 2009 or later. The tutorial starts with common implementation steps and then describes how to activate automatically or manually using **Instant SOLO Server**.

Prerequisites

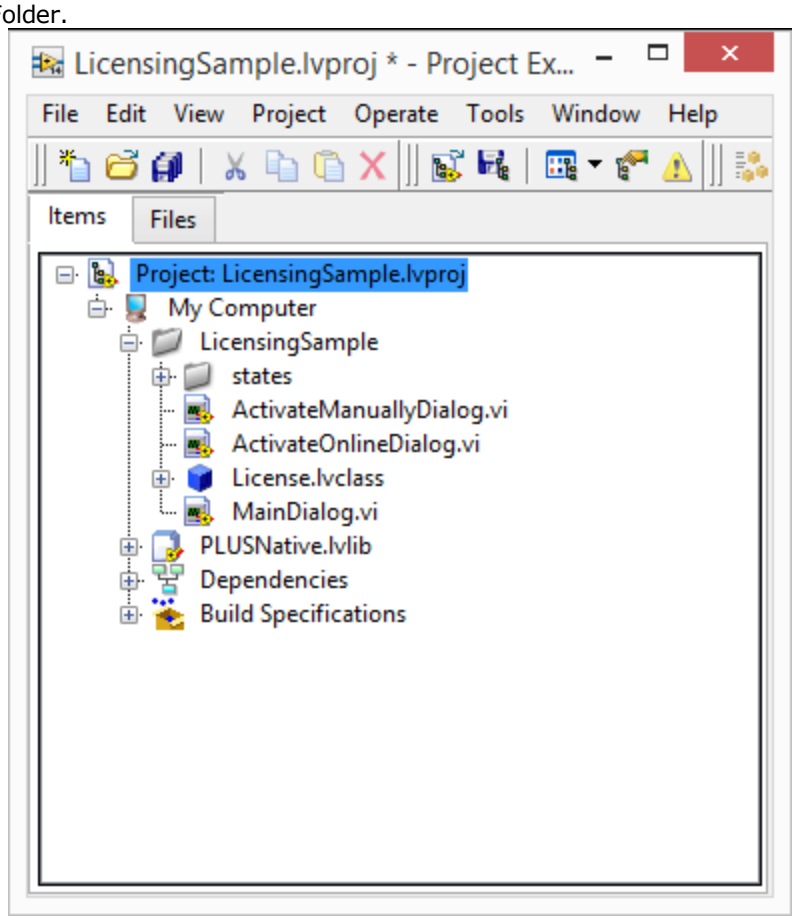
- LabVIEW 2009 or later
- "LabVIEW Executable" LabVIEW project.
- Protection PLUS 5 SDK LabVIEW Edition from Concept Software (30-evaluation available at **http://www.softwarekey.com/**)
- Internet Connection

Step 1 - Configuring the Licensing Sample Project

Open the "LicensingSample.lvproj" project found in the *Samples* directory. The *Samples* directory can be opened by running Protection PLUS 5 SDK from your start menu/screen, and clicking on the Protection PLUS 5 SDKSamples link.

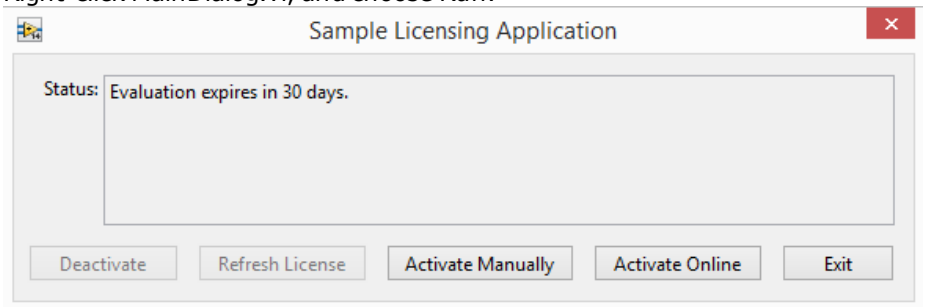


Once the Licensing Sample has loaded in the LabVIEW development environment, expand the Licensing Sample Folder.



>

Right-click MainDialog.vi, and choose Run.



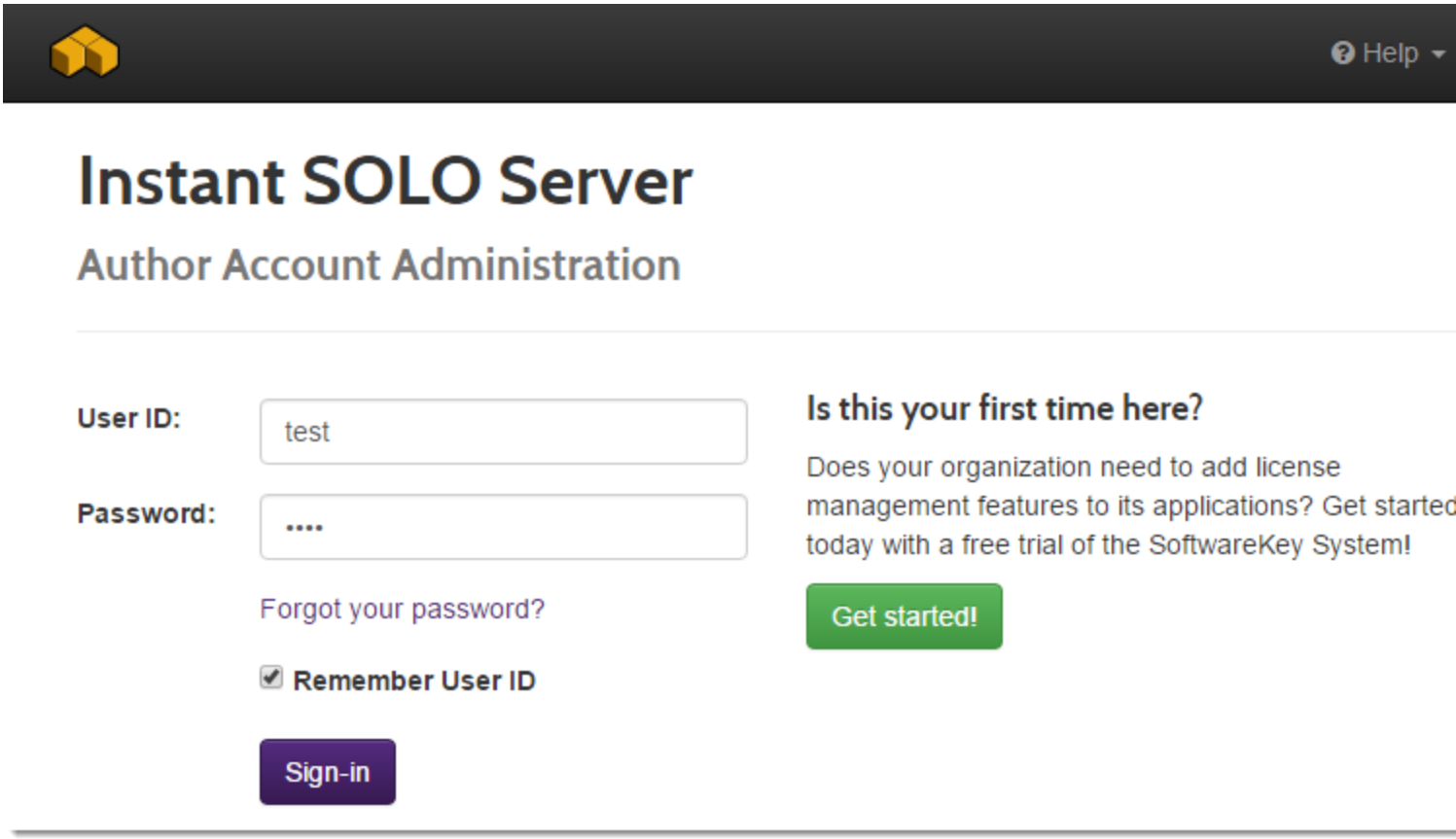
Step 2 - Activating automatically with Instant SOLO Server

To activate the Sample Licensing Application automatically using *Instant SOLO Server*, click on Activate Online. A new dialog will prompt for a License ID and Password (both are required). Specifying an installation name is optional, but helpful when performing multiple installations for different machines and/or users.

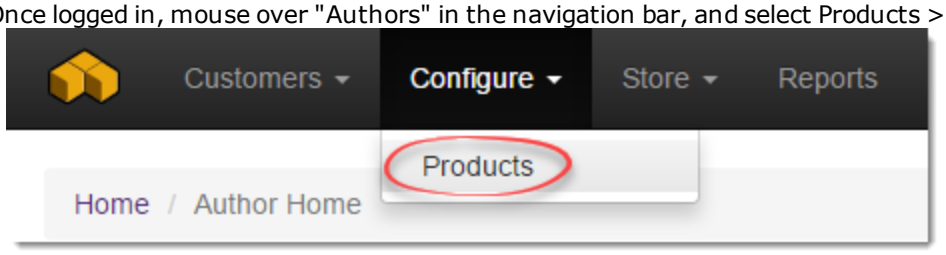
For ease of use, all of the Protection PLUS 5 SDK sample applications use a generic Test Author account with its own *Encryption Key ID*. We will use this Test Author account on SOLO Server to generate the License ID and password necessary to activate the Licensing Sample.

To log into Instant SOLO Server, visit **https://secure.softwarekey.com/solo/authors/Default.aspx**. Use the Test Author credentials given below:

- User ID: test
- Password: test

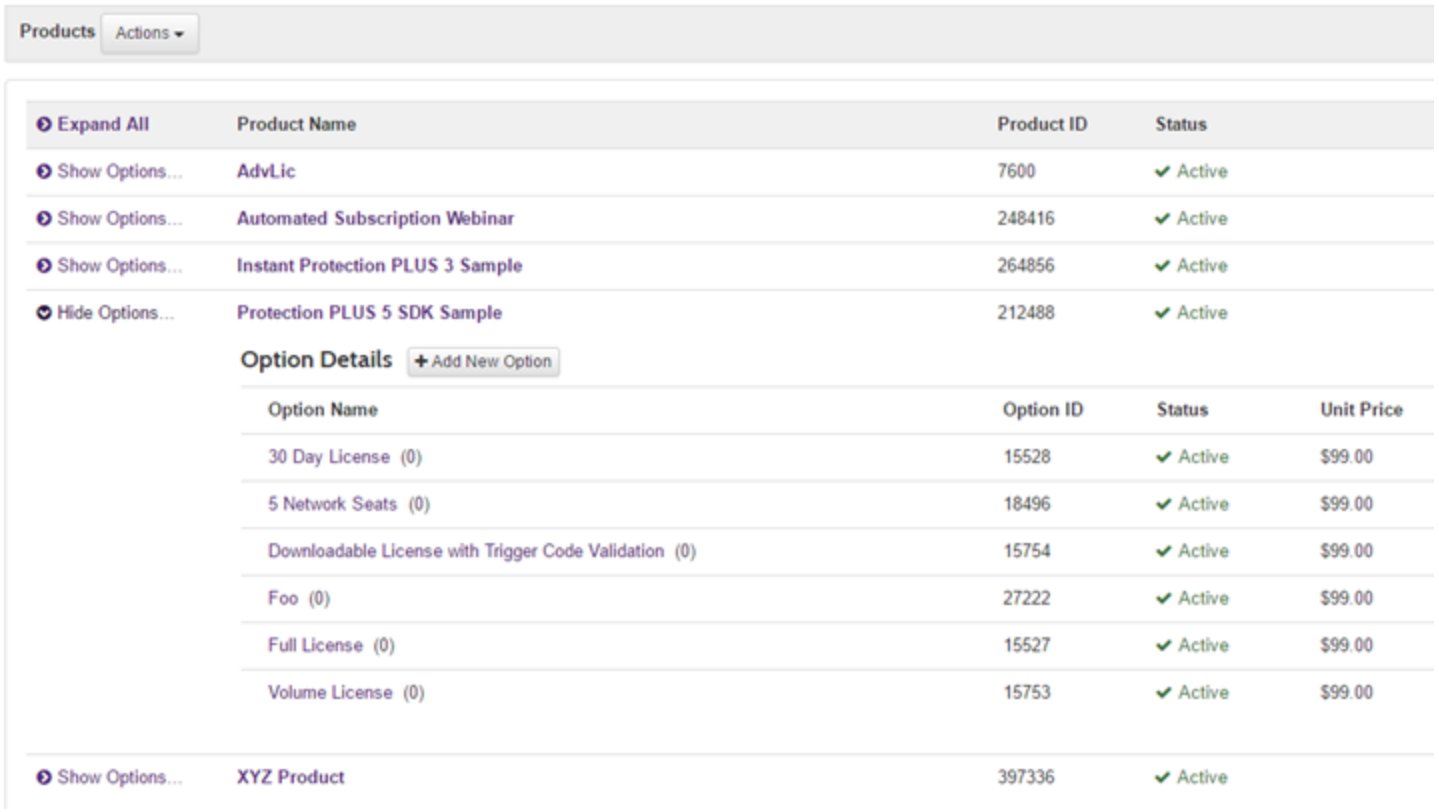


Once logged in, mouse over "Authors" in the navigation bar, and select Products > List.

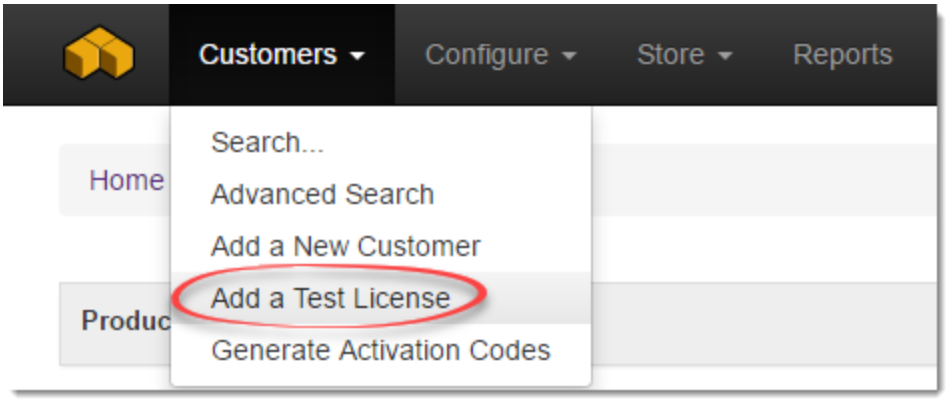


>

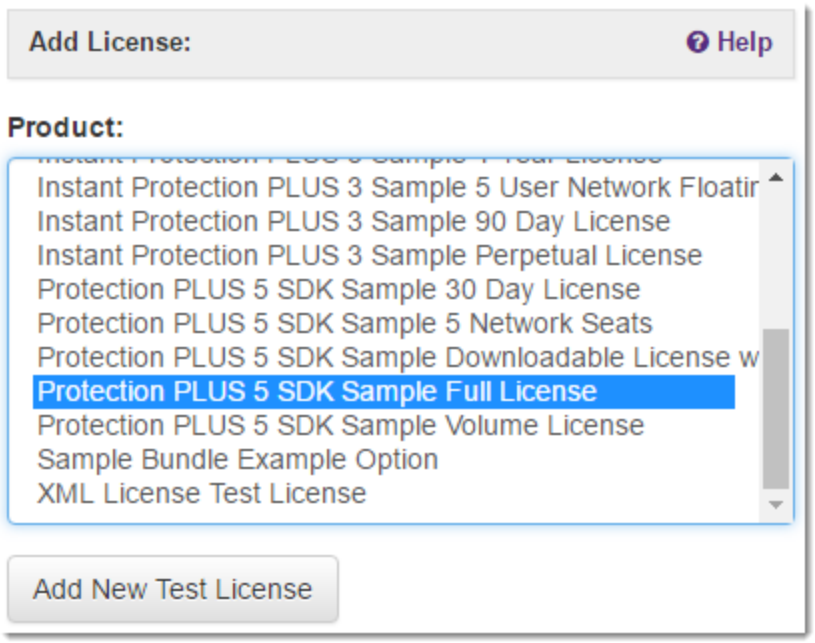
The Product List page will show all of the products available to the Test Author. Expand "Show Options" beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for activating the Licensing Sample application. For this tutorial, we will generate a license for the Full License.



To add a test license for this product option, mouse over Customers in the navigation bar, and click "Add Test License".

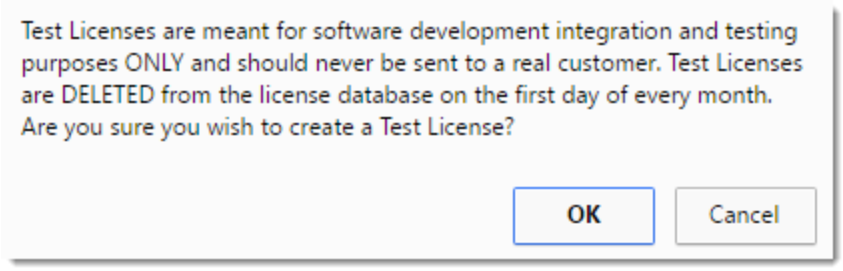


Select the Protection PLUS 5 SDK Sample Full License and click Add New Test License.



Important:

Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.

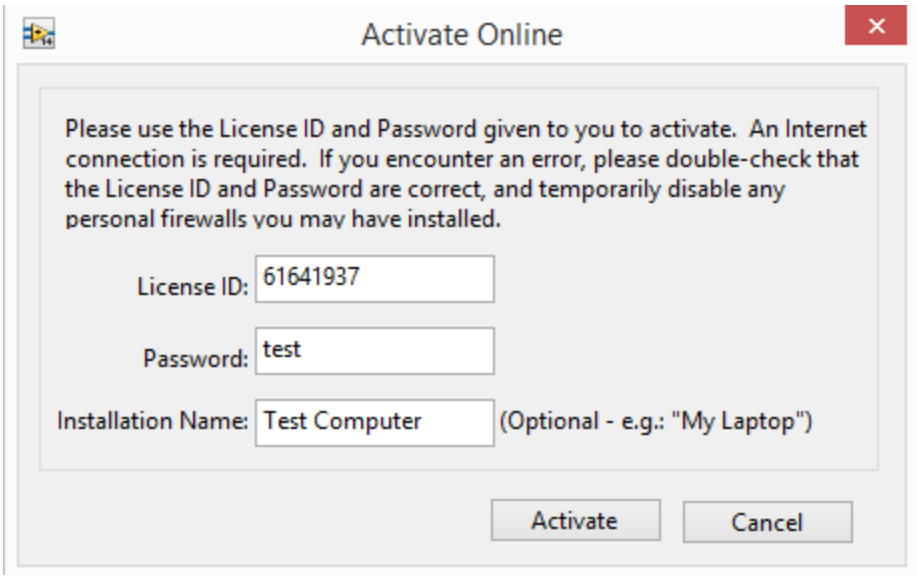


">

The Test License has now been created. We now have a License ID and Password to continue testing the Licensing Sample.

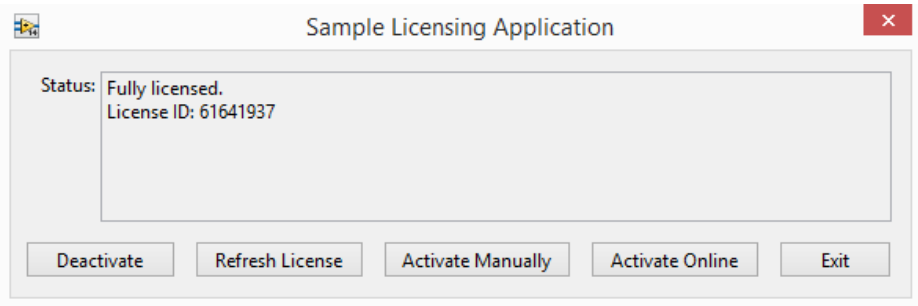
Return to the activation prompt for our Licensing Sample. Enter the License ID and Password in their respective fields and, optionally, an Installation Name. Click Activate to communicate with Instant SOLO Server for license val-

idation.



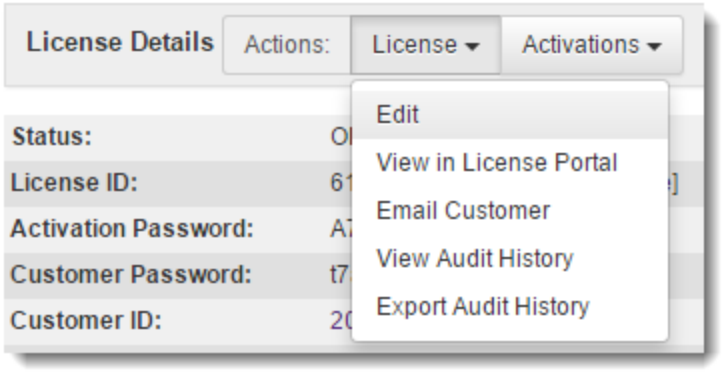
Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.

The License Status will now show as Fully Licensed.

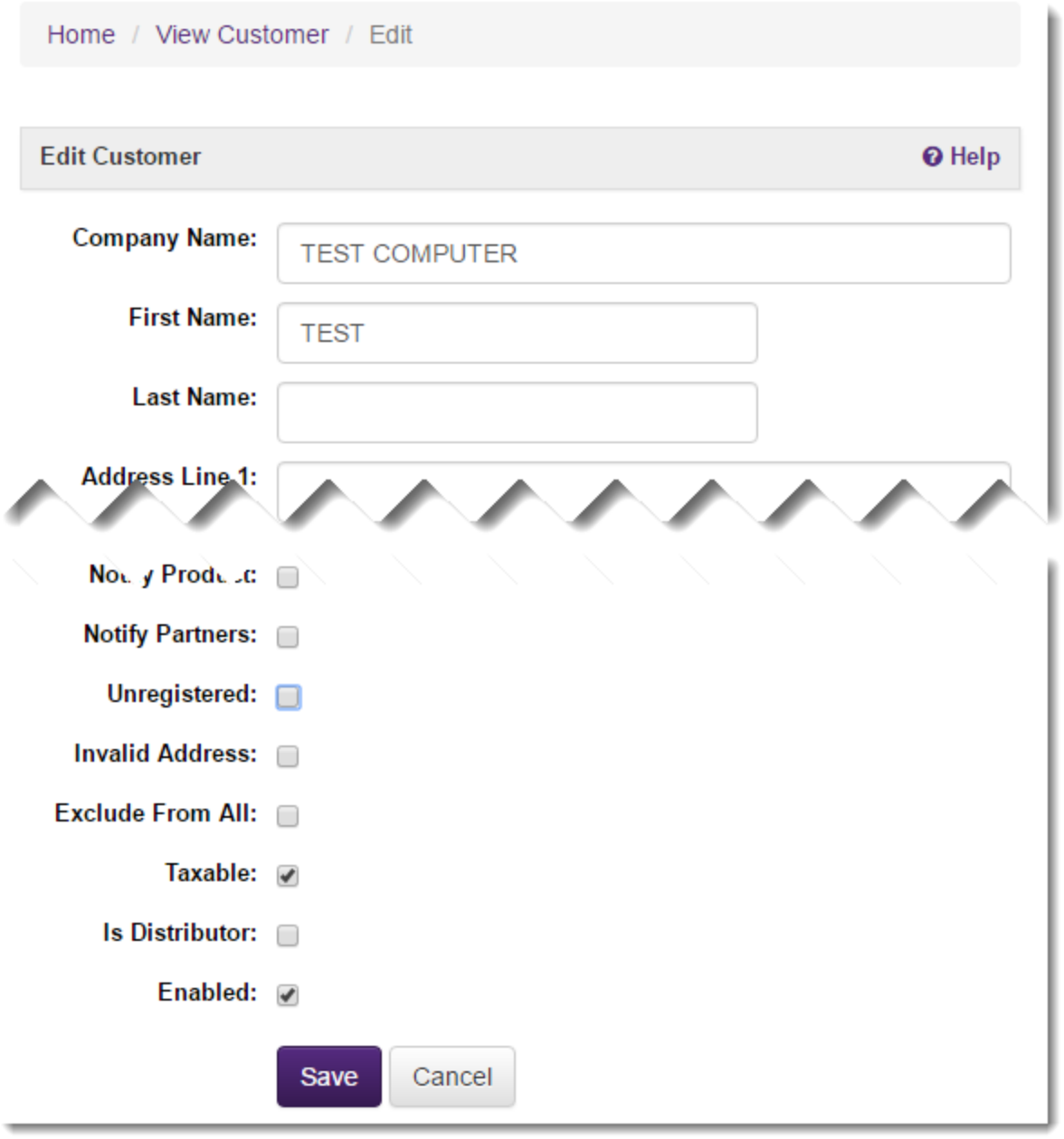


Step 3 - Refresh a License

A license can be refreshed with updated information sent from Instant SOLO Server. We can modify customer details from the License Details page on Instant SOLO Server. On the License Details page, click the License drop-down and choose Edit.



Click View Customer to get to the Customer Details page then click Edit. Modify the Company Name and additional fields as you please. Uncheck the box to the right of Unregistered. Scroll to the bottom and click Save.



After the Customer Details have been saved, the page returns to the Customer Information page. This page includes customer details we have edited in addition to licenses and previous orders.

Home / Customer Details

Customer Information: Edit | Change Password | Save Page Layout | Help

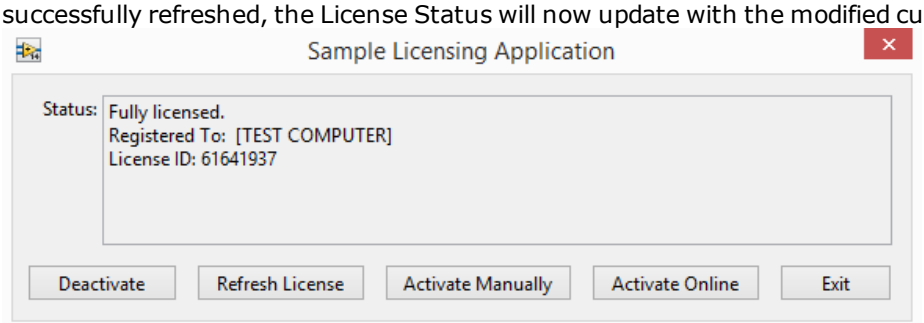
Customer ID:	20985217 [View Cust Page]
Password:	t7au2XH9
Company Name:	TEST COMPUTER
Contact Name:	TEST

Entered By:	Test
Entered Date:	11/11/2016 4:15:32 PM
Modified By:	Test
Modified Date:	11/15/2016 9:58:16 PM
Unregistered:	False
Enabled:	True
Invalid Address:	False
Taxable:	True
Exclude From All:	False
Is Distributor:	False
Notify Product:	False
Notify Partners:	False

Licenses & Other Items (1) Reset All Activations **Filter:** ALL Add License

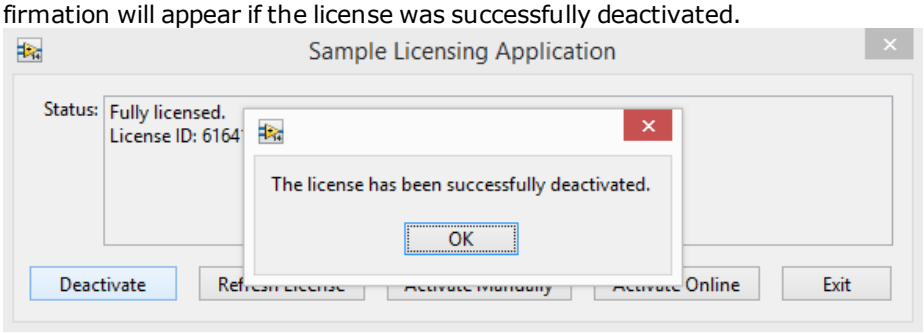
ID	Entered	Details	Qty	Expiration	Status	Last Check	Invoice
61791801	11/11/2016	Protection PLUS 5 SDK Sample Full License	1		OK A=0 D=0		

Return to the Licensing Sample Main Dialog to proceed with a license refresh. Click Refresh License. If the licensed successfully refreshed, the License Status will now update with the modified customer details.

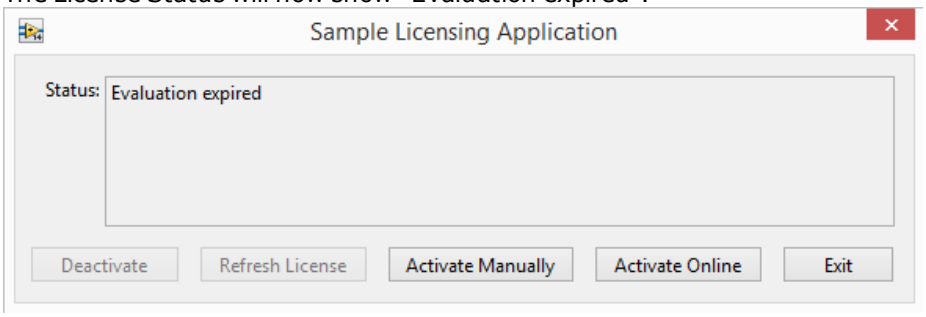


Step 4 - Process a Manual Activation

If you previously activated the Licensing Sample online using Instant SOLO Server, you will need to deactivate the license before testing manual activations. In the Main Dialog of the Licensing Sample, click Deactivate. A confirmation will appear if the license was successfully deactivated.

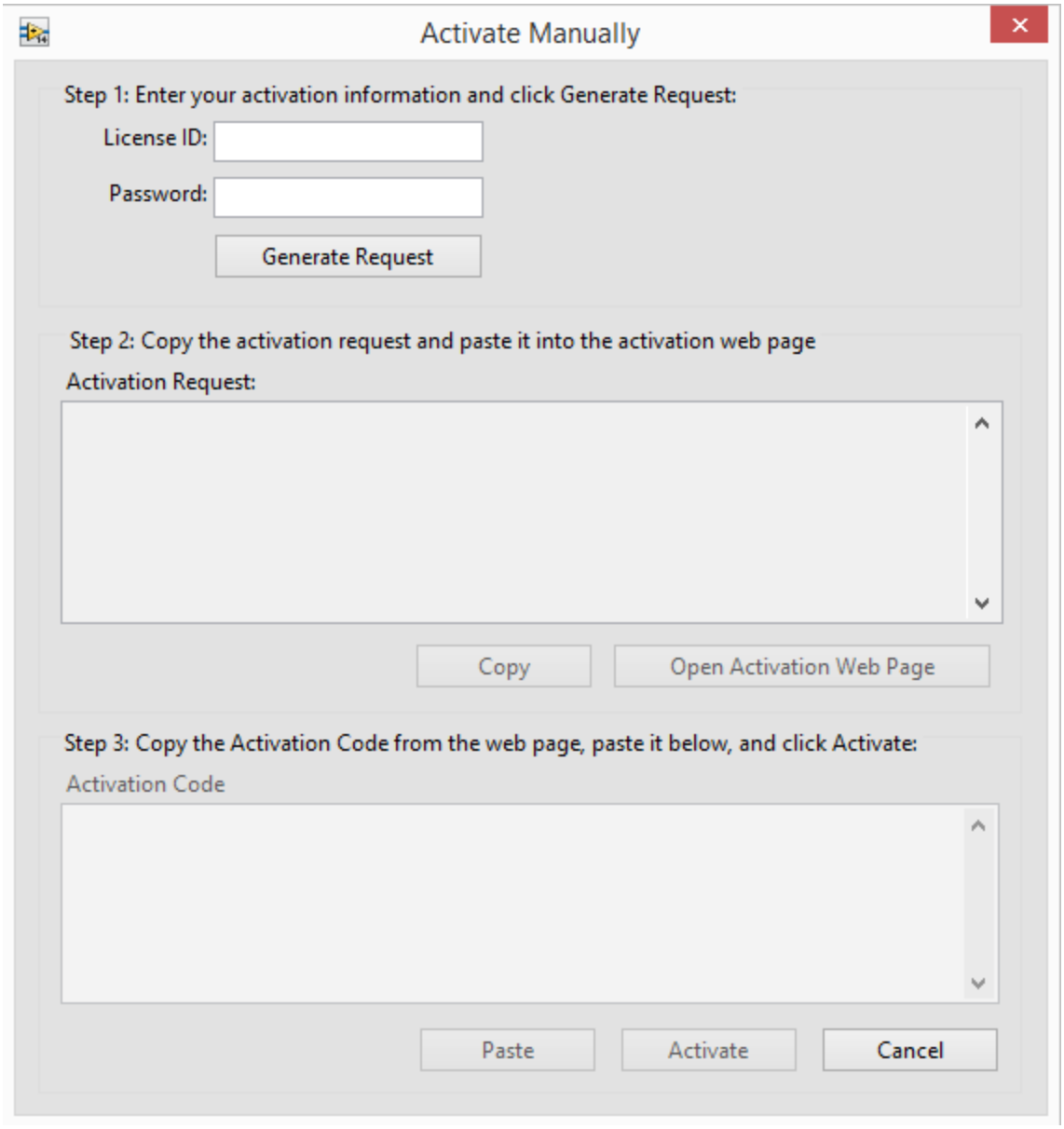


The License Status will now show "Evaluation expired".

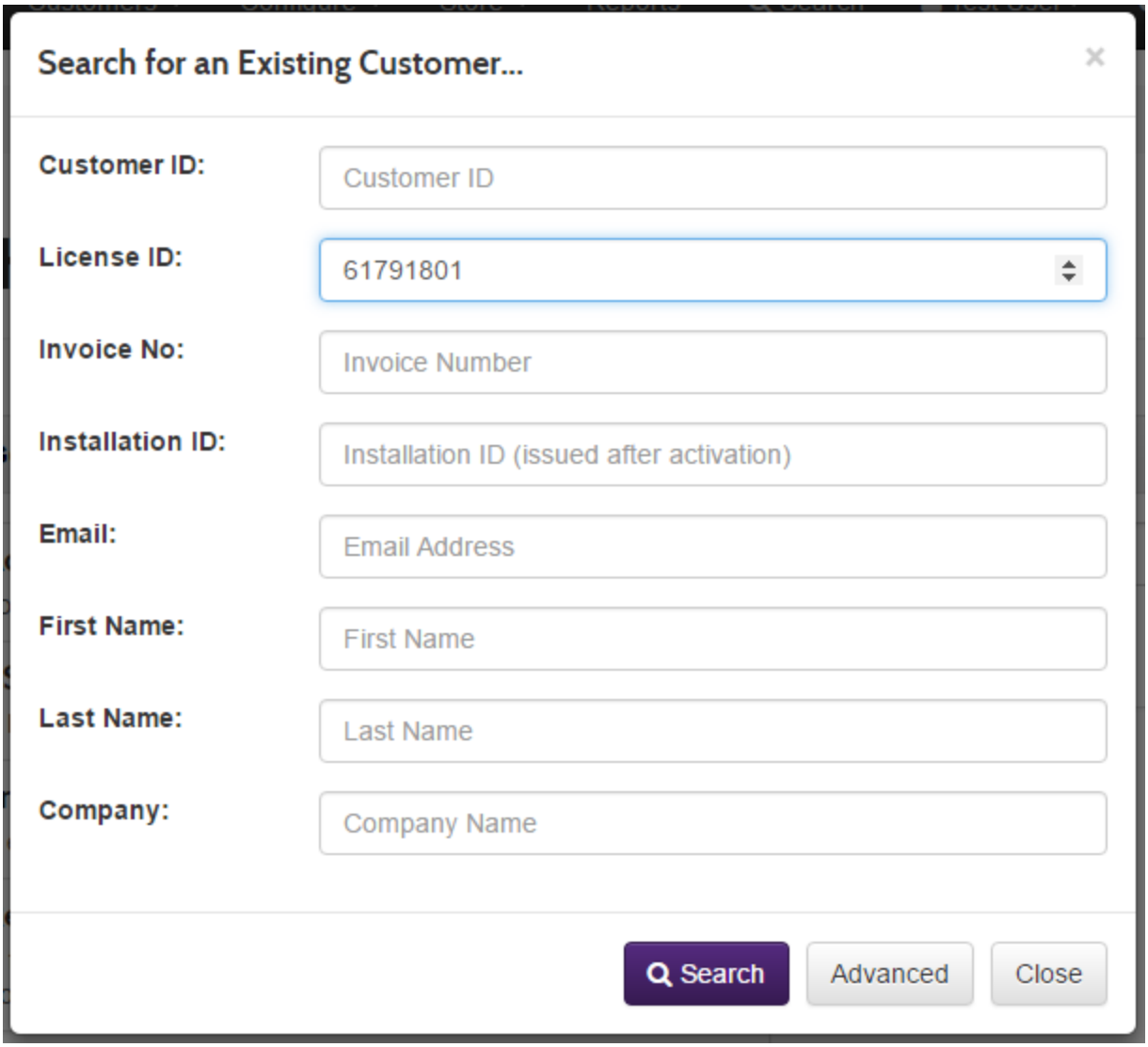


Click Activate Manually to proceed with testing a manual activation. A manual activation allows for activations from another computer or by e-mail. Follow the instructions in Step # 2 to generate a new license. Alternatively, we can also check the number of activations remaining for the License ID generated when we tested automatic activations online using Instant SOLO Server.

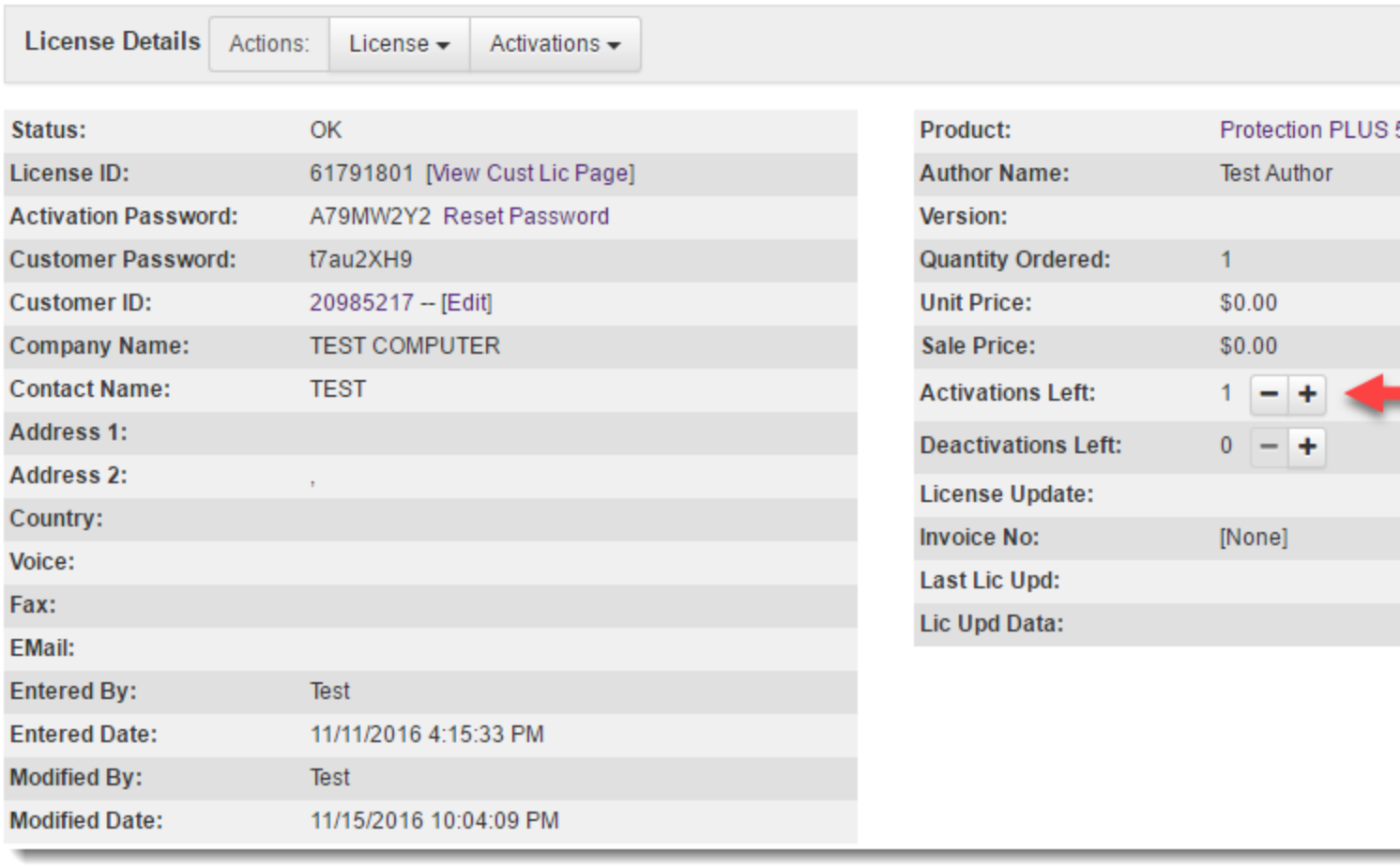
>



Return to Instant SOLO Server using the Test Author account. Perform a quick search for the License ID previously used to activate online (If you no longer have this number, proceed to create a new License ID as shown in Step # 2).

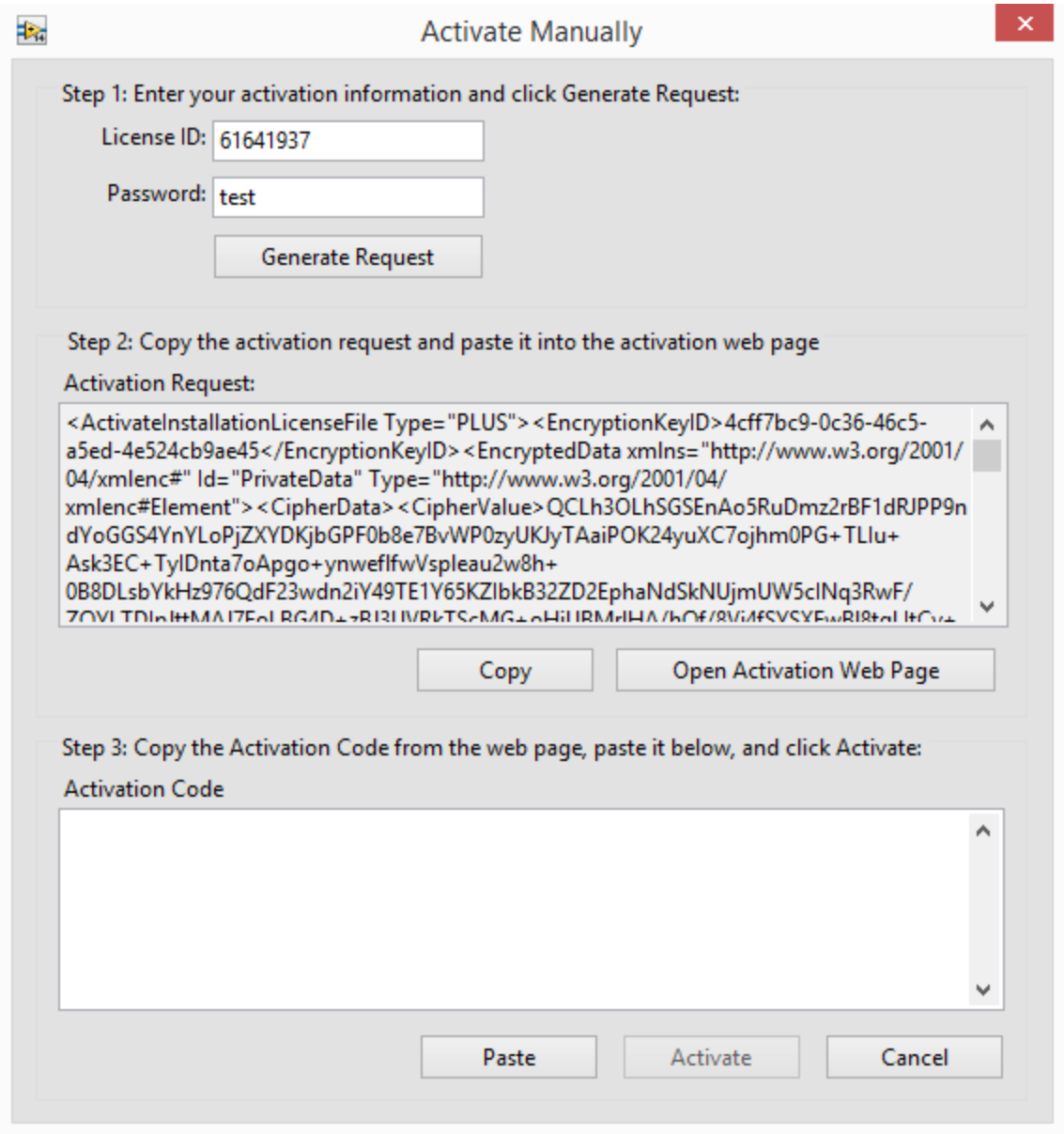


When we clicked on deactivate license, Instant SOLO Server automatically incremented the amount of activations left by 1. This setting was predefined in the Product Option ID we selected for testing. If you need to manually increase the number of activations left, you can increment this number by clicking the + button to the right of the Activations Left field.

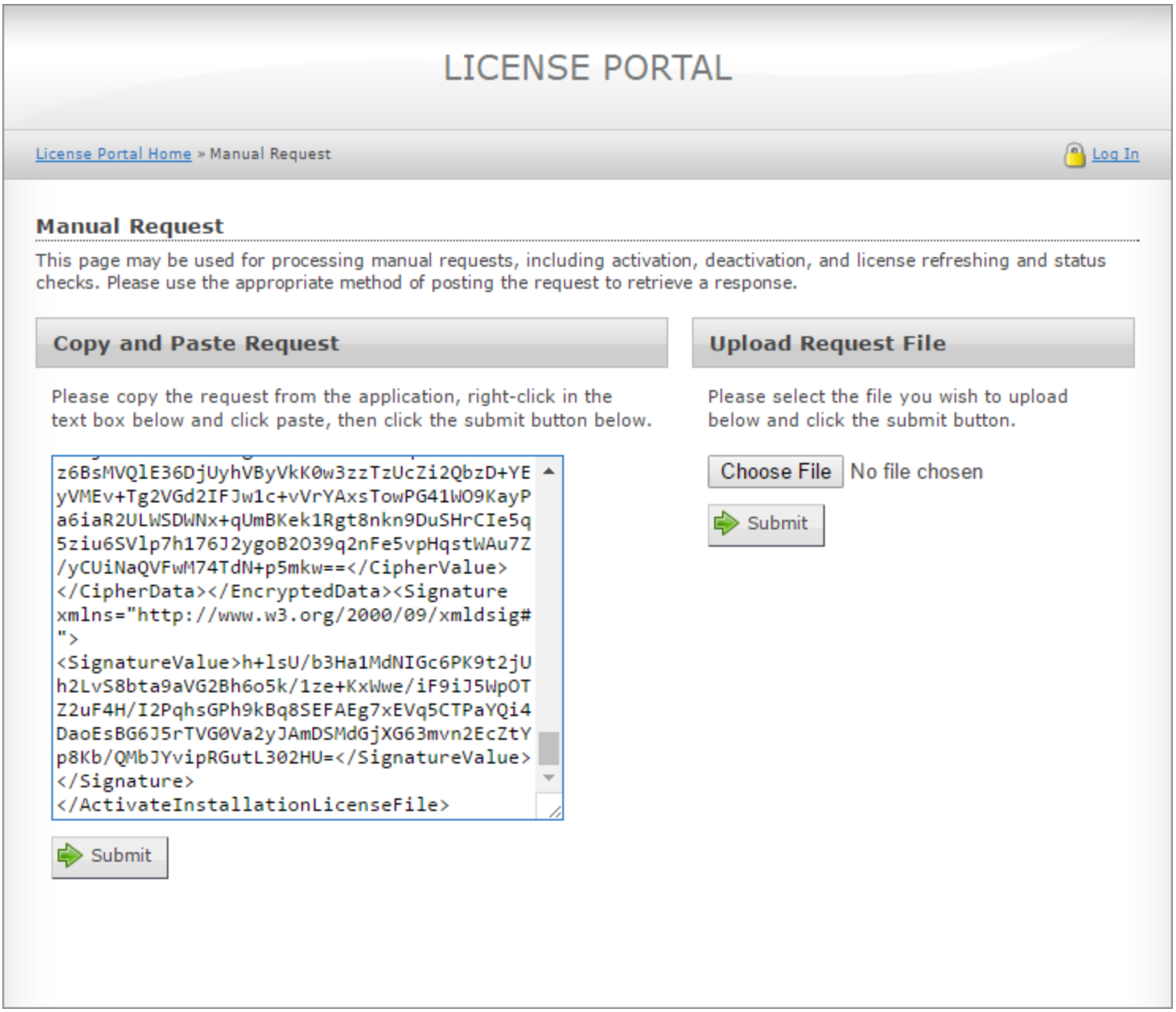


Return to the Activate Manually for our Licensing Sample. Enter the License ID and Password in their respective fields and click Generate Request.

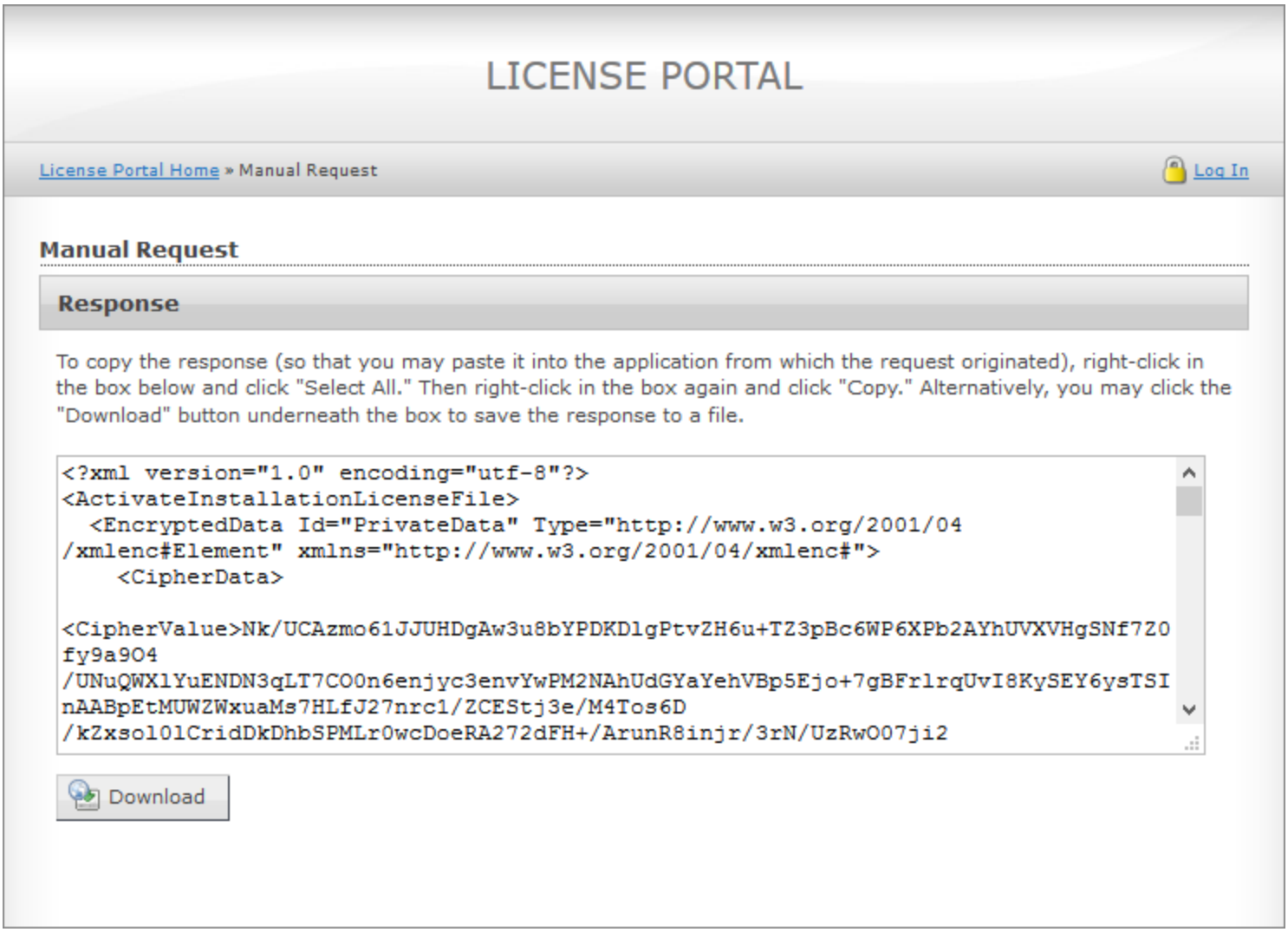
>



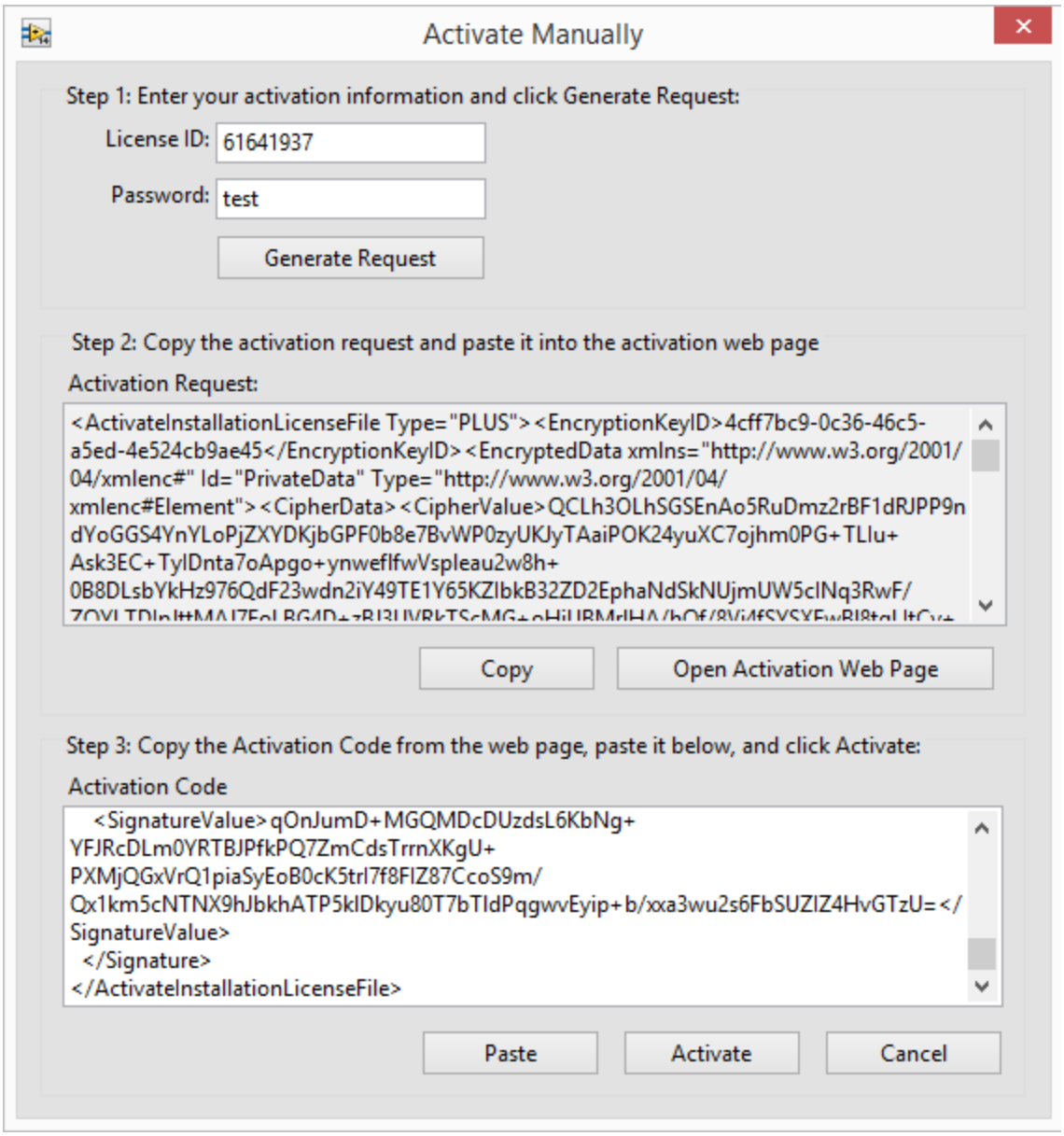
This dialog will generate an encrypted activation request to be processed by the activation server. Click Copy to copy this data to the clipboard. Next, click Open Activation Web Page. This web page is accessible from another computer with access to the Internet in the event that an offline workstation needed to be activated. Once you arrive to the License Portal page, paste the encrypted activation request data, and click Submit.



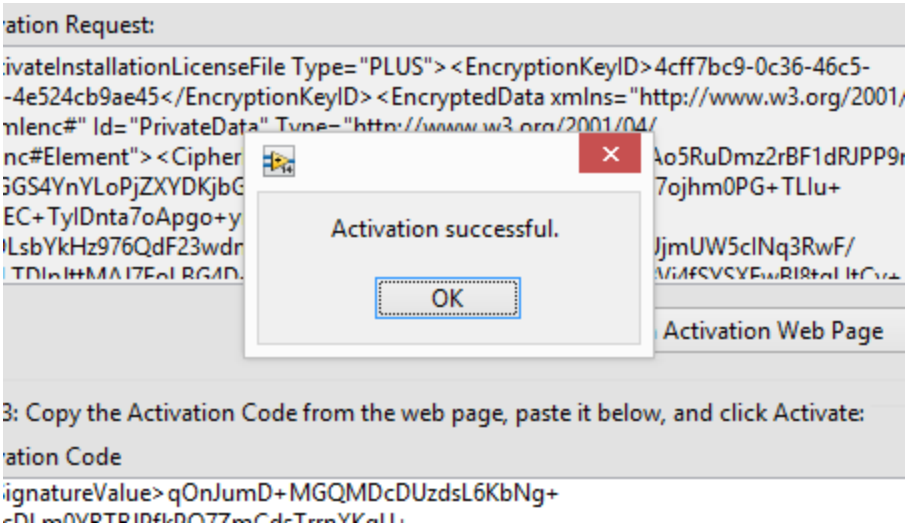
The License Portal will now generate an encrypted response. Copy this data to later paste in the Activation Code field of the activation window.



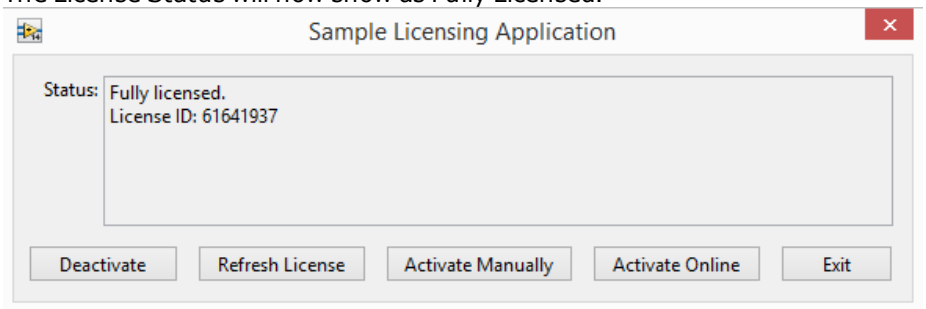
Return to the activation window and paste the encrypted response into the Activation Code field and click Activate.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



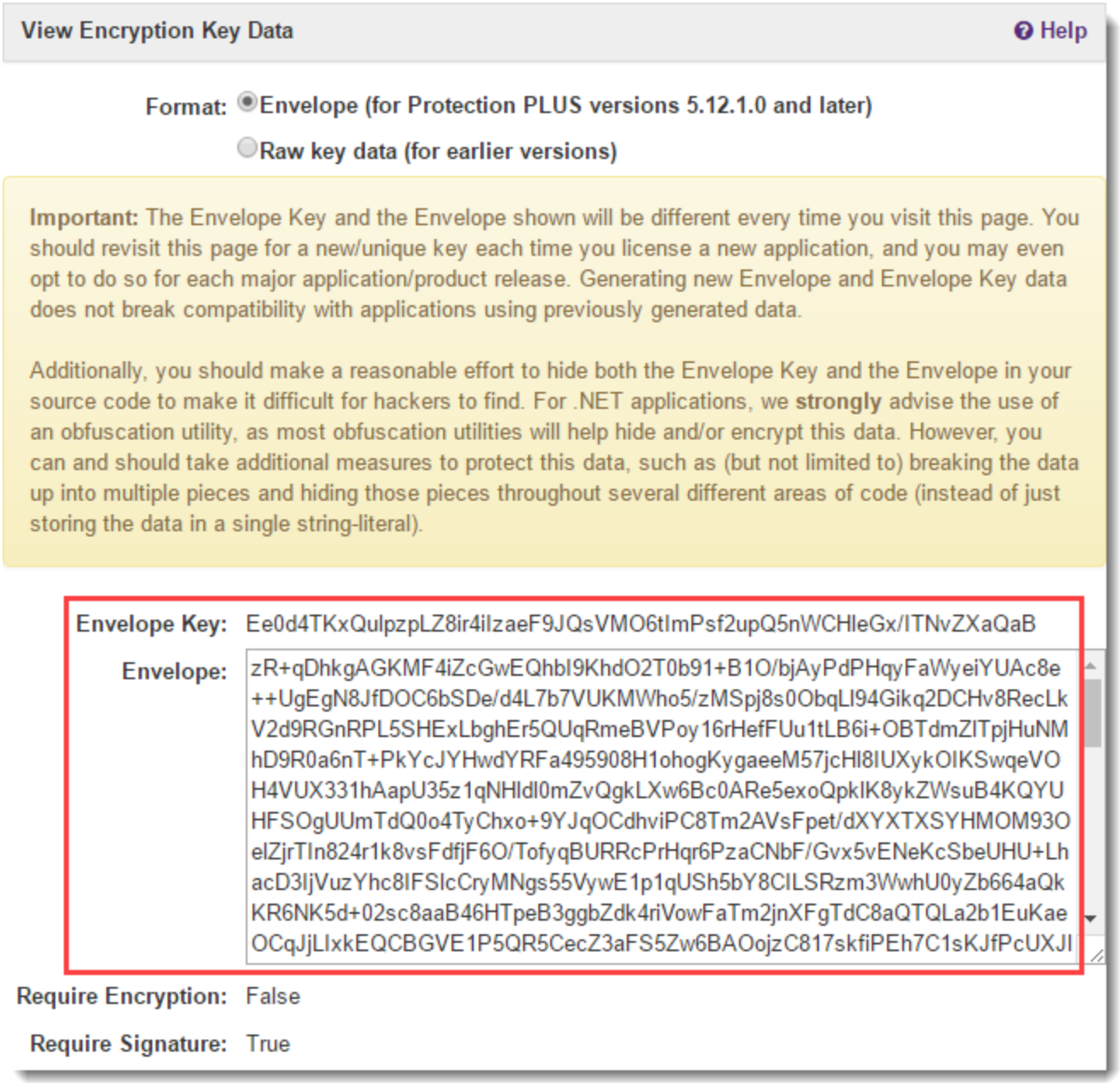
The License Status will now show as Fully Licensed.



LabVIEW Licensing Sample with your own Instant SOLO Server account

This tutorial will demonstrate how to protect your application using the Protection PLUS 5 SDK and LabVIEW 2009 or later. The Protection PLUS 5 SDK has many API functions designed to help you implement basic and advanced licensing features. Review the Understanding Licensing Topic and the Common Implementation Steps for a general understanding of how to implement licensing techniques.

For simplicity, we will be using the LicensingSample project to help get you started. You are free to use or customize the License class in the LicensingSample in your LabVIEW project. We encourage you to create a new class to call API functions that satisfy your licensing needs. See the **Defining Licensing Requirements topic**.



Next, create a new Product using the instructions in the **Instant SOLO Server Product Configuration topic**. Go to the menu *Configure / Products* then use the Actions dropdown and choose *Add*. When you are finished configuring your product, scroll to the bottom and click Submit.

Your new product will now appear in the Product List. Click Show Options next to your product to expand the Option Details. Click Add New Option to configure your application's activation settings. Give your product option a name, such as "Full License". For this tutorial, we will create a license that fully activates the LicensingSample, it will not be necessary to change any additional settings. Scroll down to the bottom and click Submit.

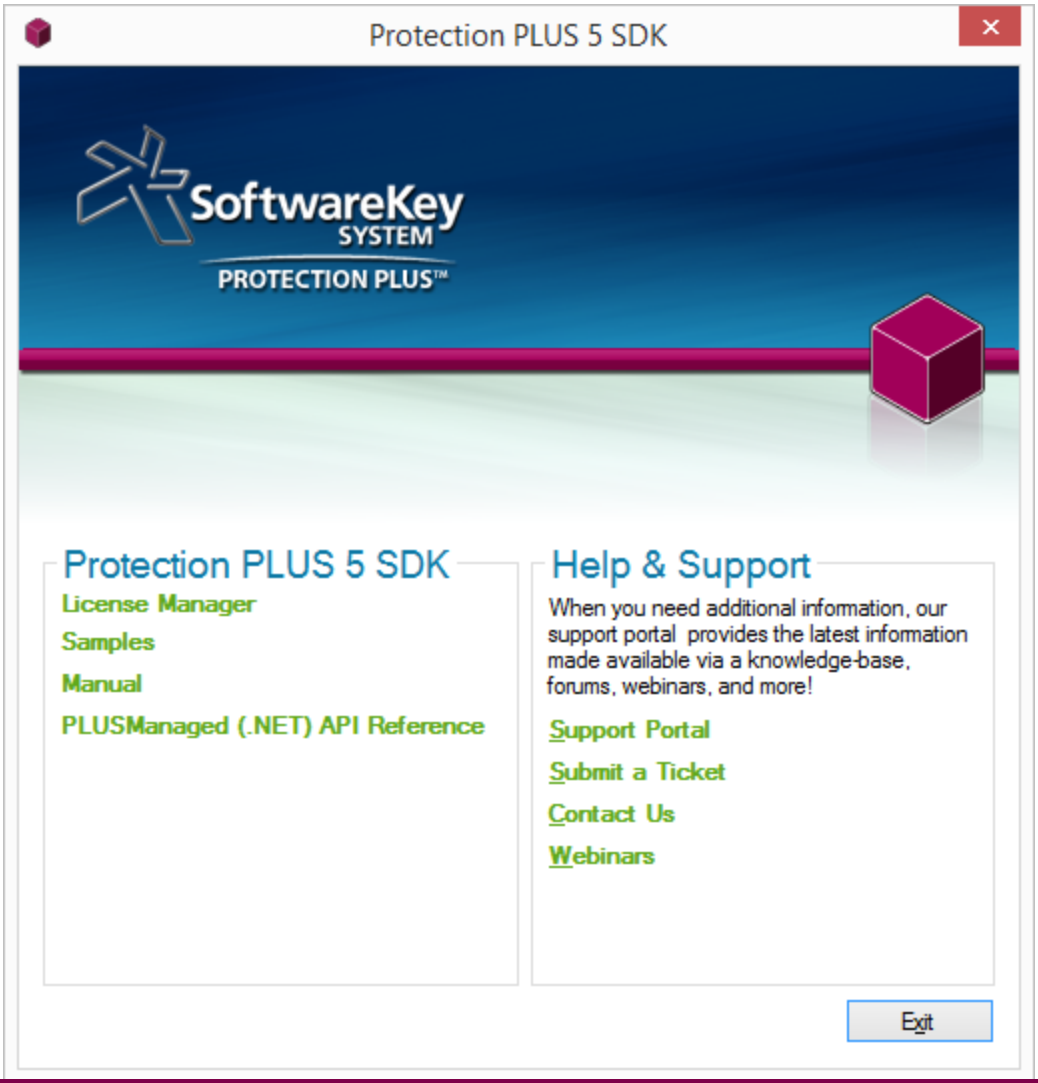
Important:

If you choose to specify a Minimum Activation Version for your product, this number cannot be greater than the version we will later specify in the application. The Minimum Activation Version field represents the minimum version number of your application that your customer must have installed to permit activation. Leave this blank to allow activation for any and all versions of your software. The format of the version number is NNNNN.NNNNN.NNNNN.NNNNN, where N represents a non-negative integer ranging from 0 to 99999, inclusive. No leading zeroes are required. Please note that all four version elements must be specified. For version numbering schemes using less than four version elements, use trailing zeroes (e.g., for version 3.0, specify 3.0.0.0).

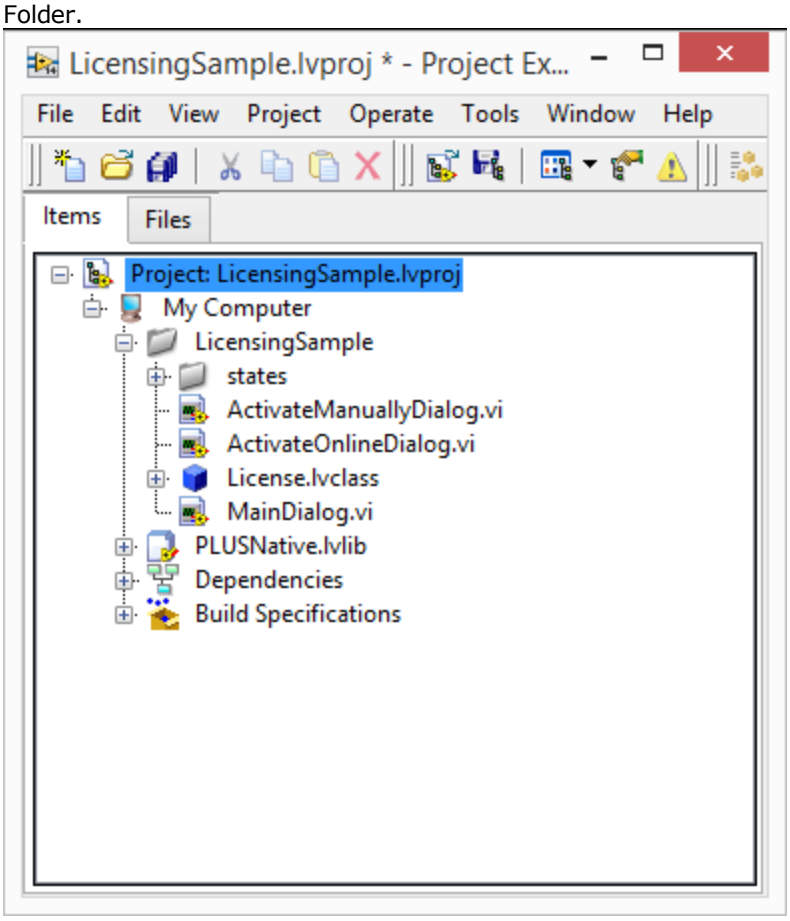
We are now ready to start integrating this information into your LabVIEW application.

Integration

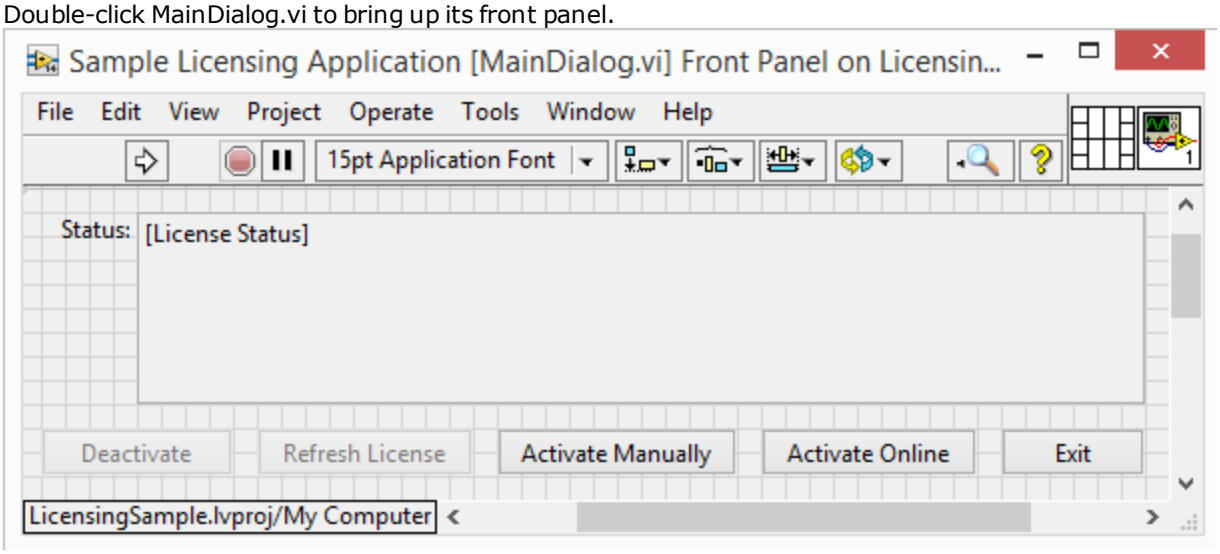
Open the "LicensingSample.lvproj" project found in the Samples directory. The Samples directory can be opened by running Protection PLUS 5 SDK from your start menu/screen, and clicking on the Protection PLUS 5 SDK Samples link.



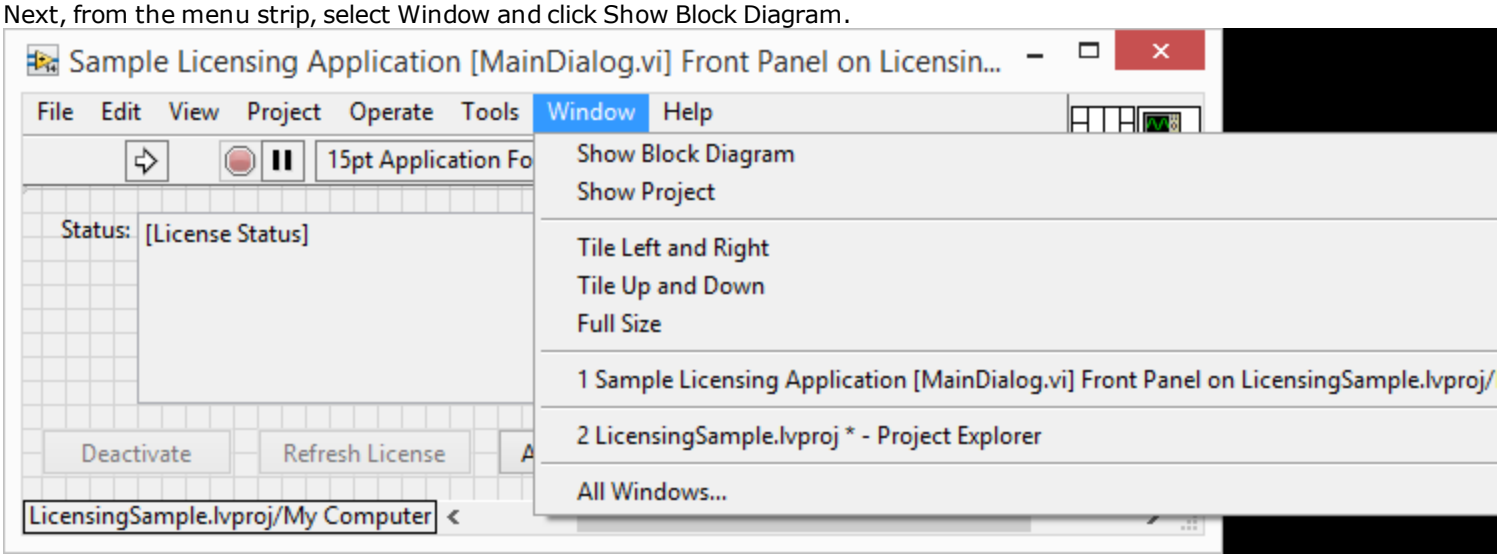
Once the Licensing Sample has loaded in the LabVIEW Development Environment, expand the Licensing Sample Folder.



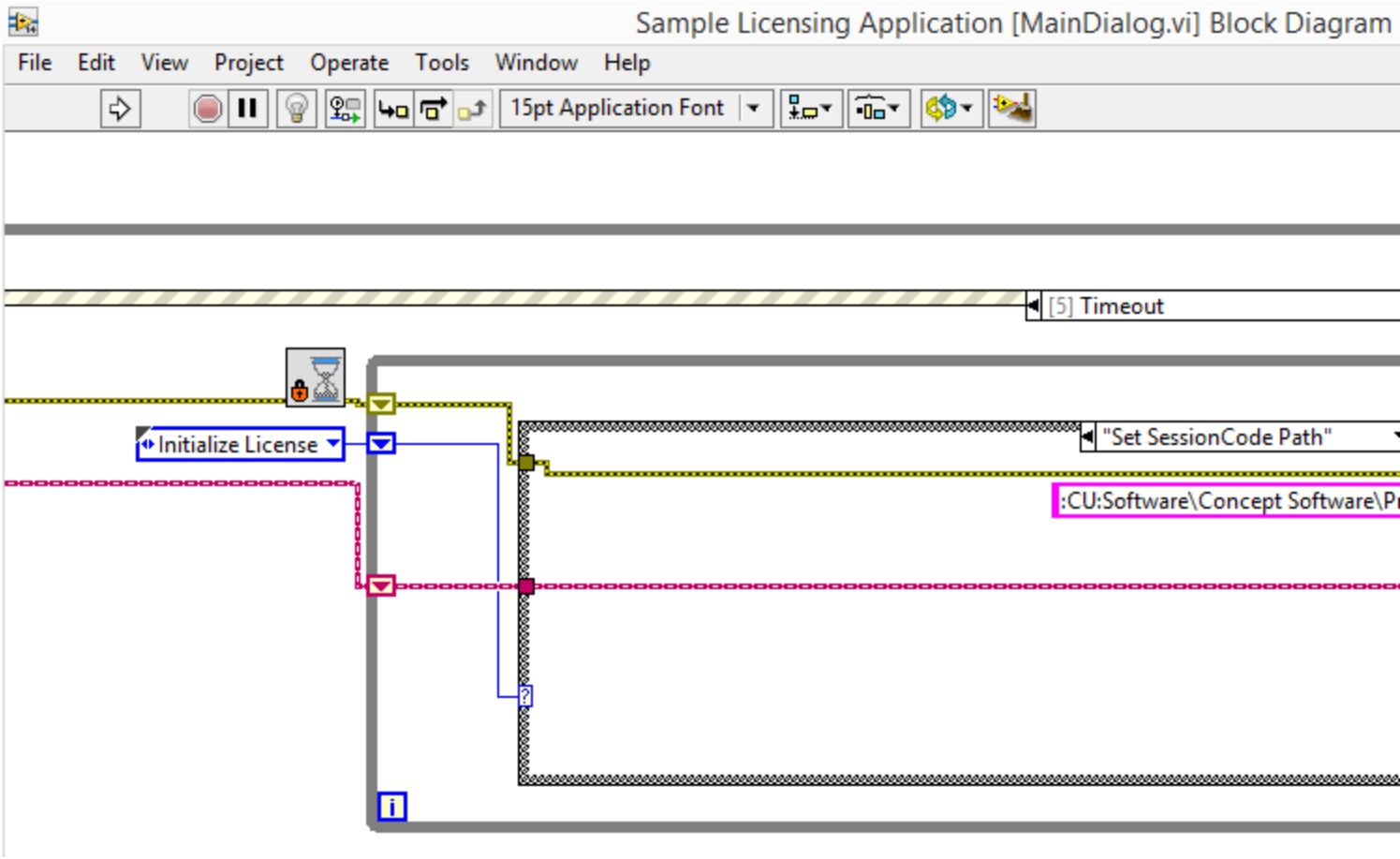
Double-click MainDialog.vi to bring up its front panel.



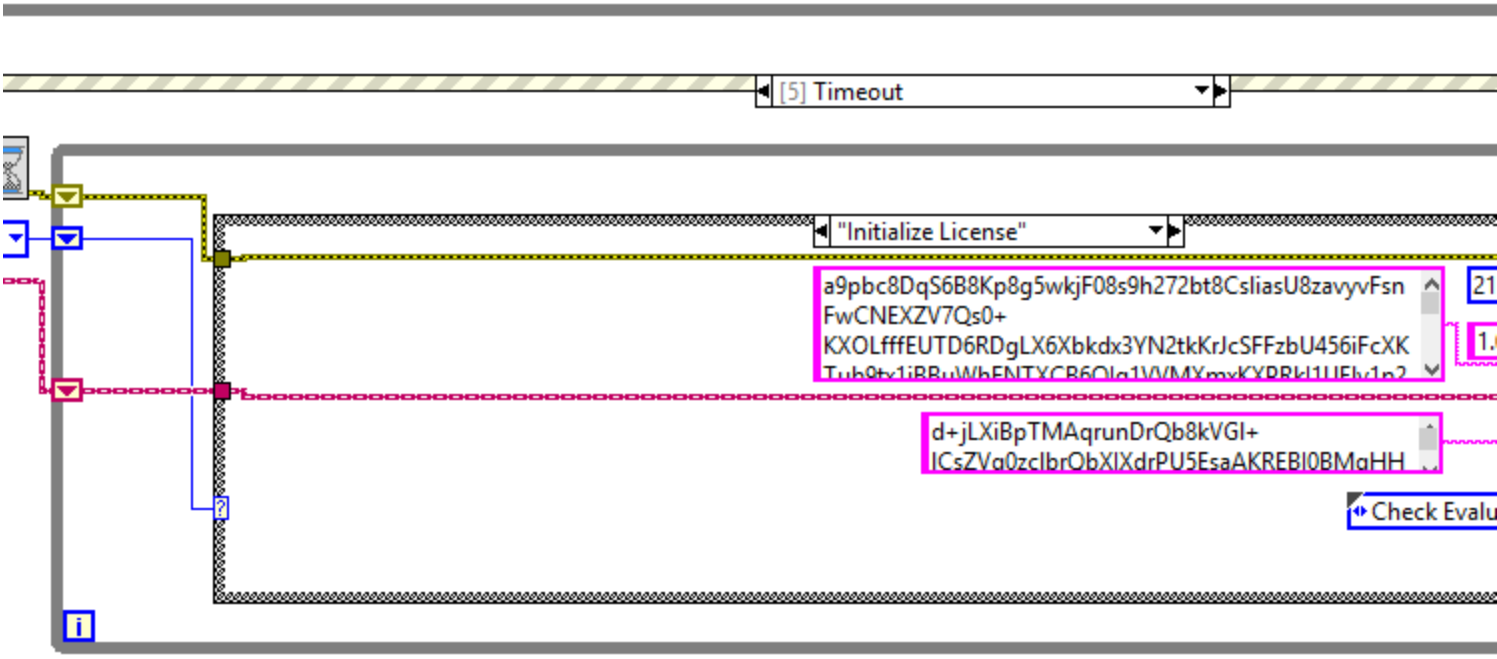
Next, from the menu strip, select Window and click Show Block Diagram.



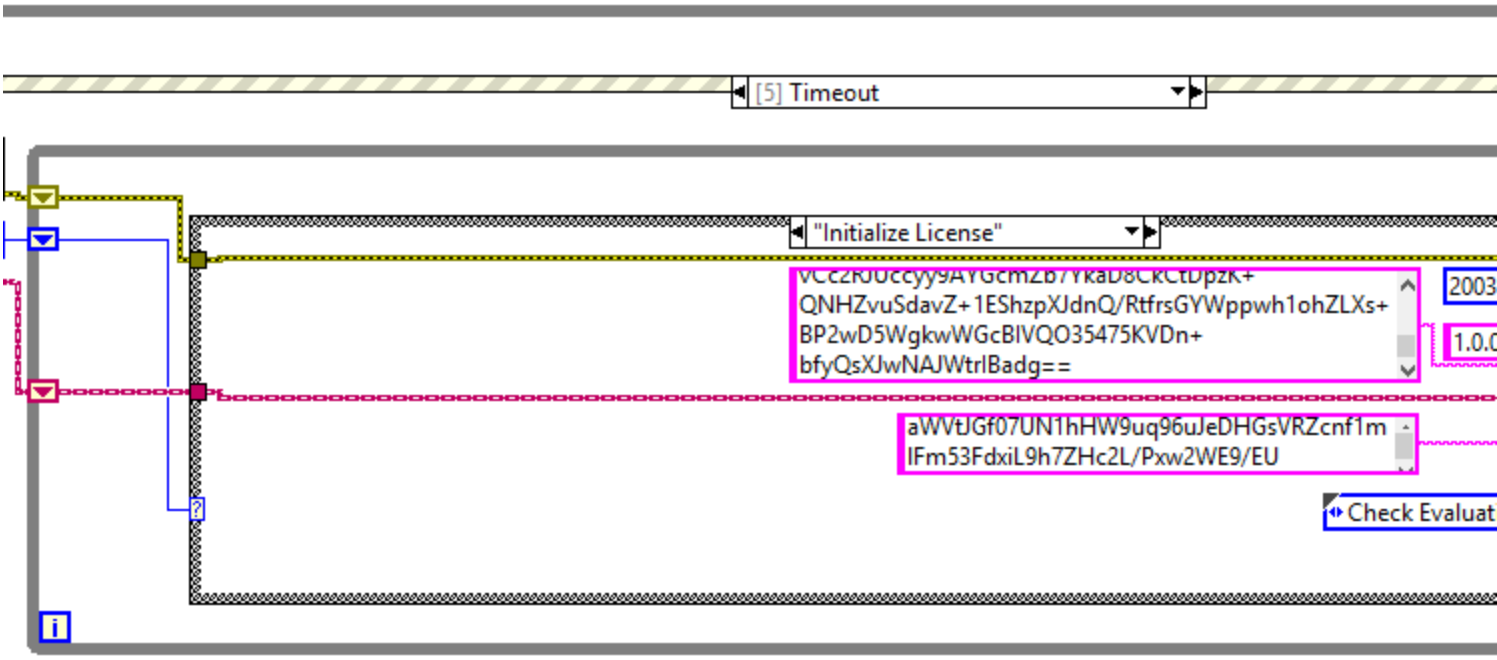
Once the Block Diagram has loaded, we will need to make some modifications to the source code in order to ensure that the application matches the product options we specified on Instant SOLO Server.



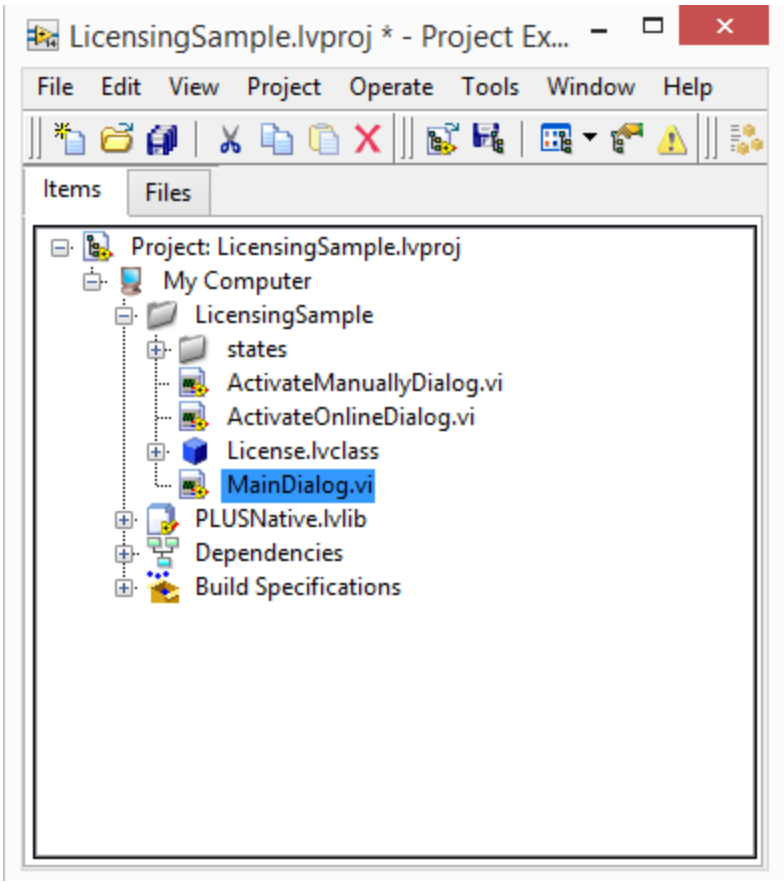
The case structure on the center of the diagram has a drop down selection currently set to "Set SessionCode Path". Click the drop down arrow to the right and select "Initialize License". We will edit several necessary fields to properly enforce licensing.



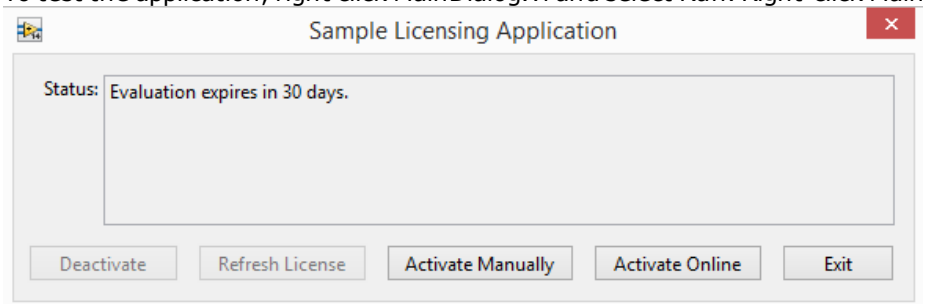
The large text box contains the Envelope we will need to replace using the Envelope we generated on SOLO Server in step #___. You will also need to replace the data in the text box below with the Envelope Key from Instant SOLO Server. The textbox containing "212488" is the Product ID. Replace this number with the Product ID of the product we created on Instant SOLO Server. The text box below is the application's version number. This number will need to be greater than or equal to the Minimum Activation Version in the Product Options on Instant SOLO Server, if you specified a value. If you left this field blank in Instant SOLO Server, any version of your application can be activated with that the Product ID option you configured. You will still want to specify your application's version number in the block diagram.



Save your changes to the MainDialog VI. Close the Block Diagram and return to your Project Files view.

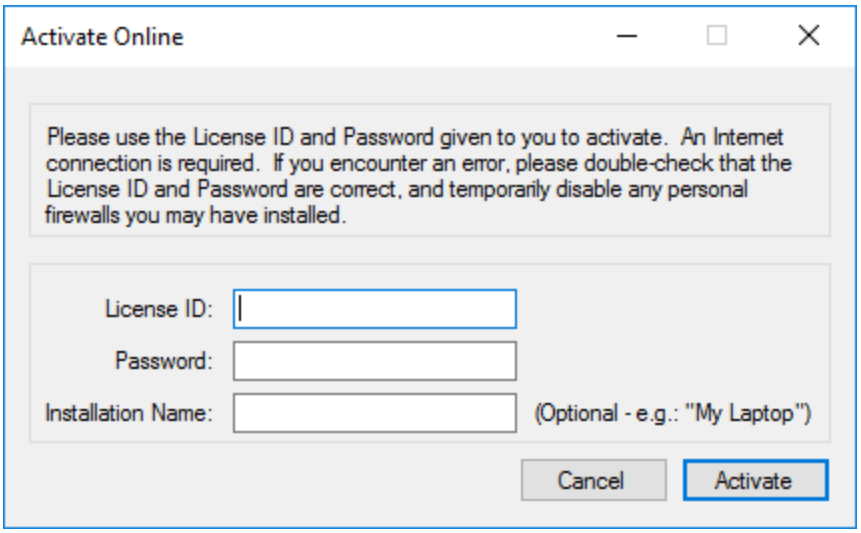


To test the application, right click MainDialog.vi and select Run. Right-click MainDialog.vi, and choose Run.

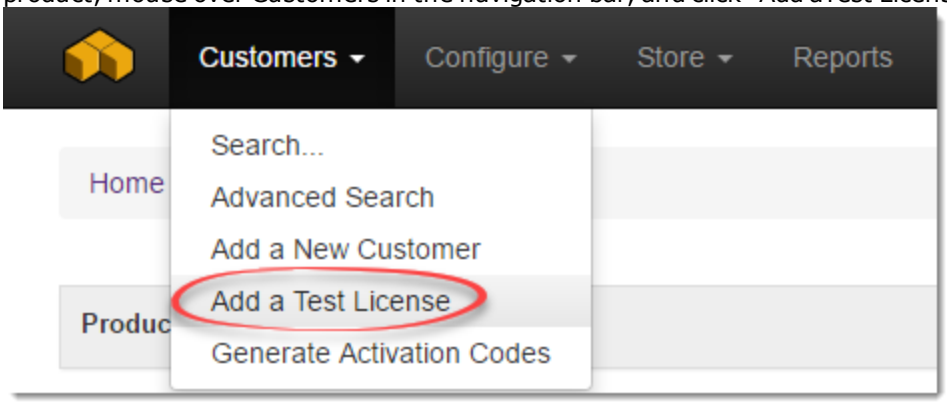


Activating automatically with Instant SOLO Server

To activate the Sample Licensing Application automatically using Instant SOLO Server, click on Activate Online. A new dialog will prompt for a License ID and Password (both are required). Specifying an installation name is optional, but helpful when performing multiple installations for different machines and/or users.



Return to SOLO Server to create a test license for the Product ID we created earlier. To add a test license for this product, mouse over Customers in the navigation bar, and click "Add aTest License".



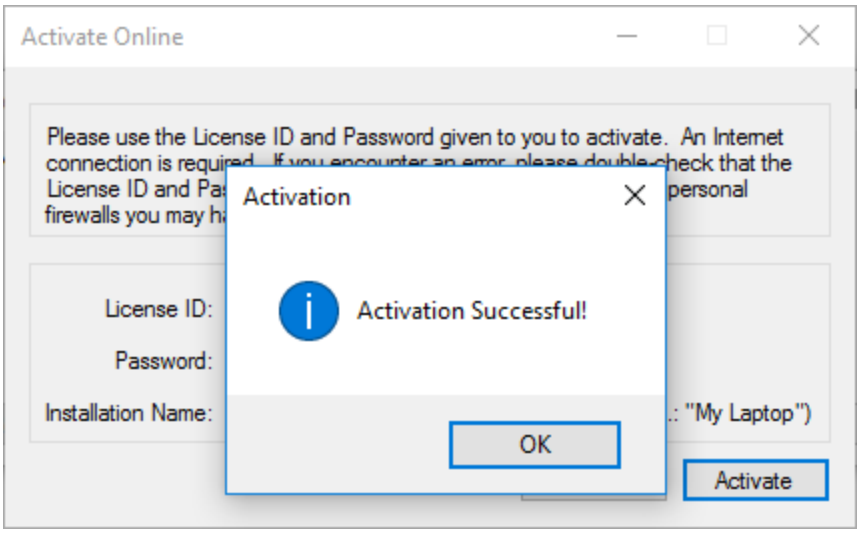
Find your product in this list and click Test License.

Important:

Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.

The Test License has now been created. We now have a License ID and Password to continue testing the Licensing Sample.

Return to the activation prompt for our Licensing Sample. Enter the License ID and Password in their respective fields and, optionally, an Installation Name. Click Activate to communicate with Instant SOLO Server for license validation. Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.


Activate Online
Please use the License ID and Password given to you to activate. An Internet
connection is required. If you encounter an error, please double-check that the
License ID and Pas
personal
firewalls you may ha
License ID:
Password:
Installation Name:
.: "My Laptop")
Activate
Activation
Activation Successful!
OK

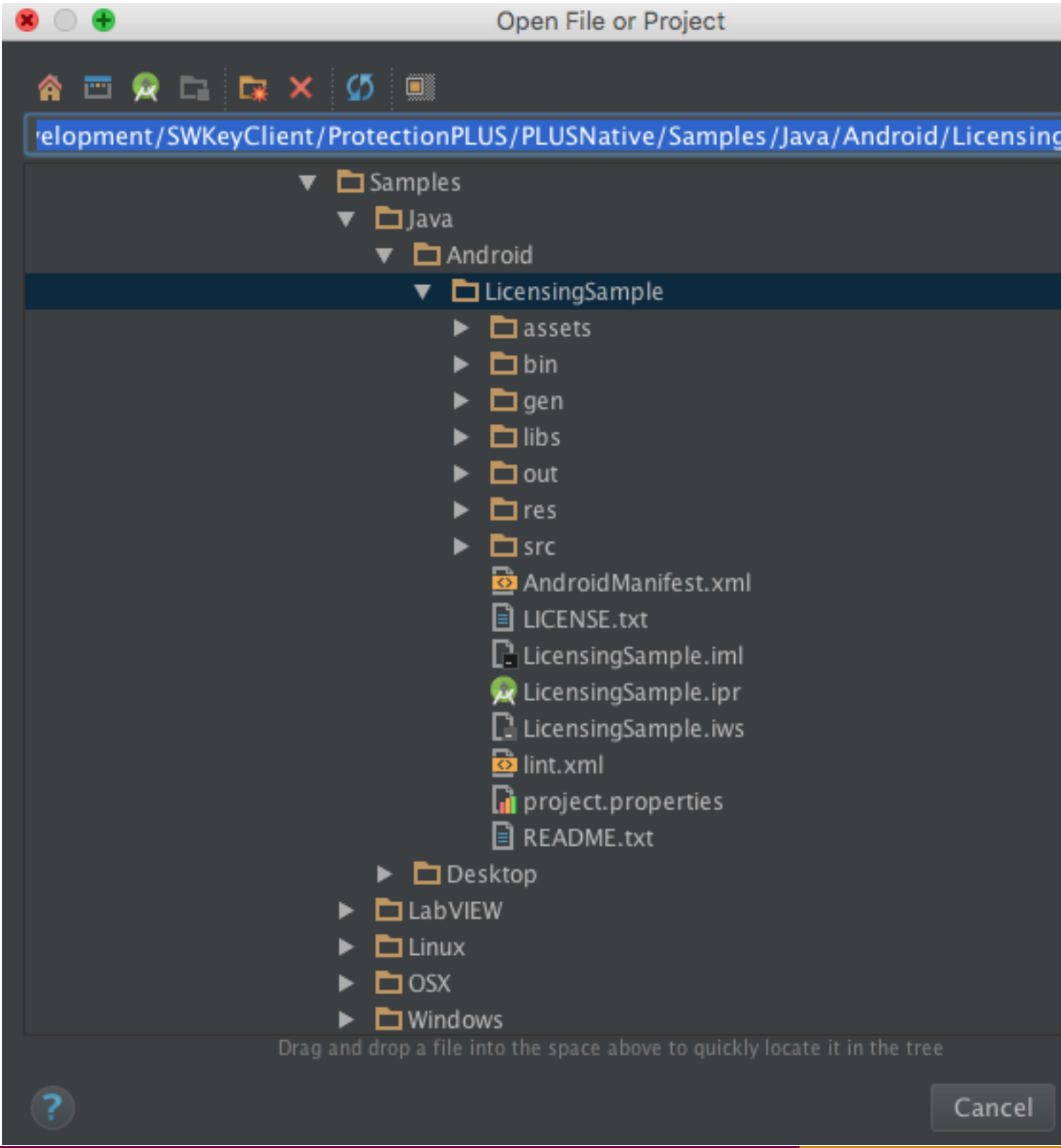
PLUSNative Android Studio Licensing Sample

Step 1 – Opening the Licensing Sample Project

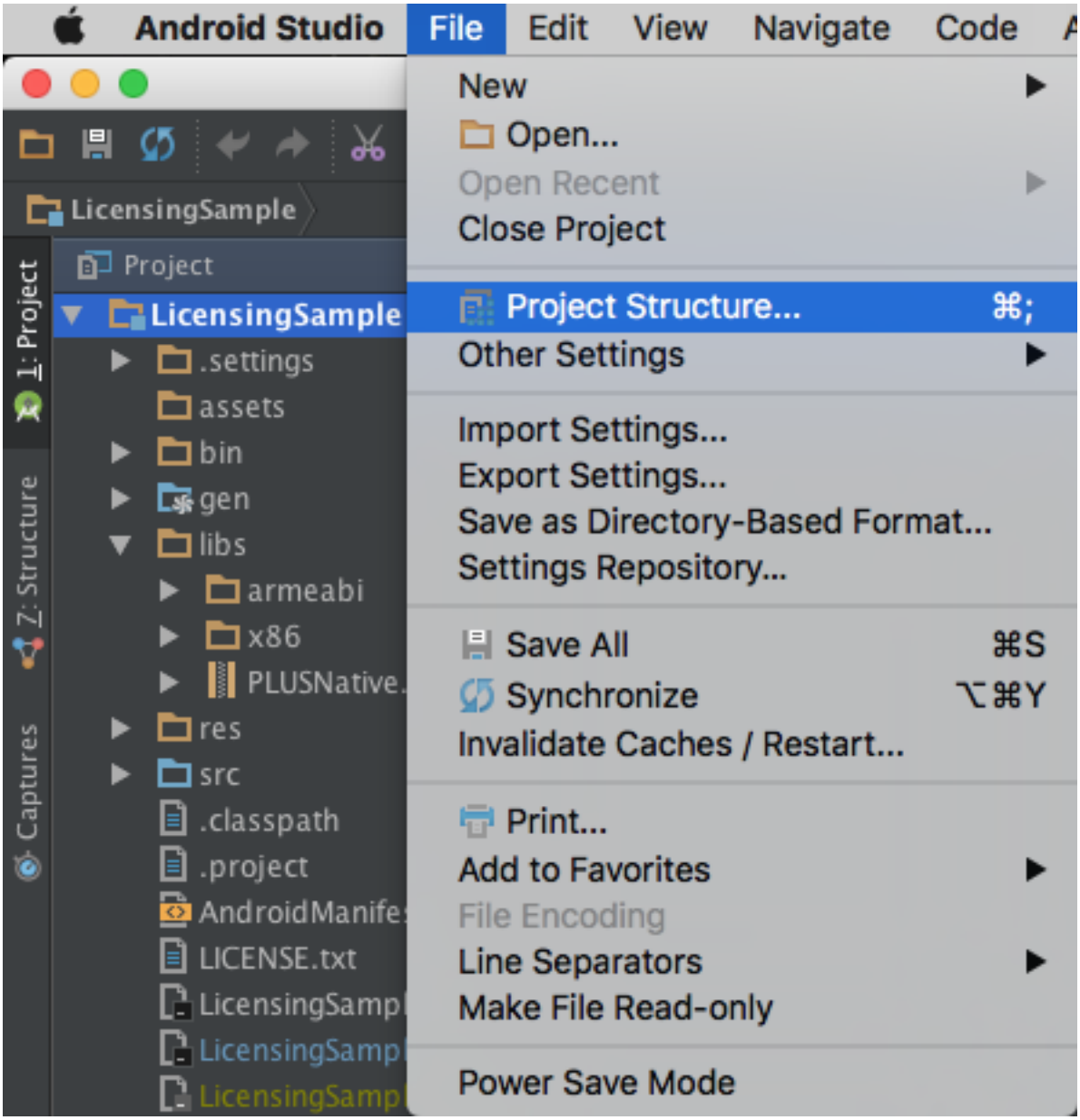
If you haven't already done so, download the Protection PLUS 5 SDK Android Edition zip file (usually named ProtectionPLUS5Android.zip). You will typically acquire a link or button to download this zip file in an email if you are evaluating, or by logging-in as an existing customer at **www.softwarekey.com**. After downloading this zip file, extract its contents to a directory / folder of your choosing, and take note of the location you selected.

Next, open **Android Studio**. From the Welcome screen select "Open an existing Android Studio Project", or from within Android Studio use the File/Open menu. Browse to the *Protection PLUS 5 SDK Android Edition\Samples\Java\Android\LicensingSample* sub-directory located where you extracted the Protection PLUS 5 SDK Android Edition zip file.

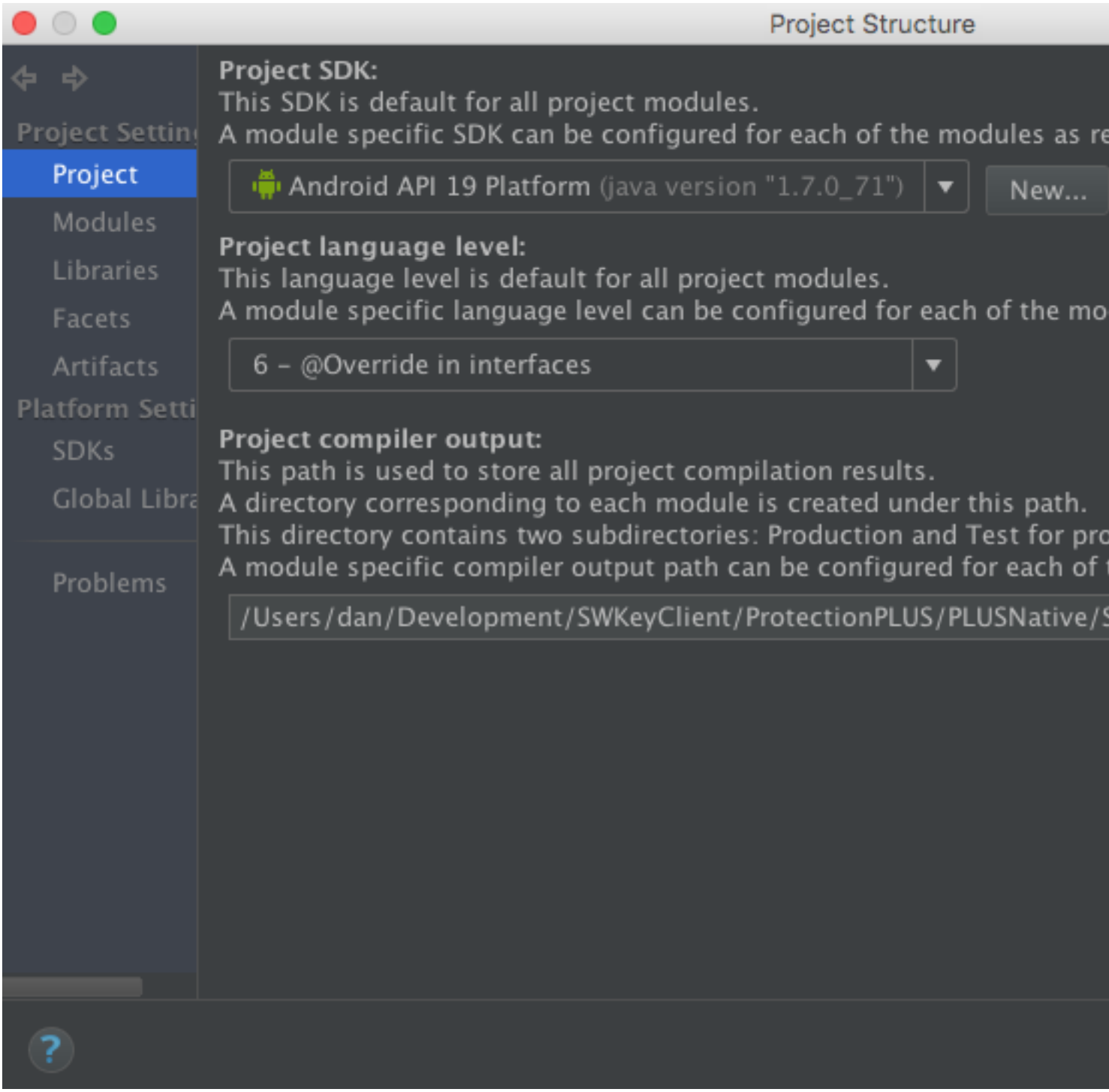

Welcome to Android Studio
Android Studio
Version 2.1.2
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Import an Android code sample
Configure


Open File or Project
elopment/SWKeyClient/ProtectionPLUS/PLUSNative/Samples/Java/Android/Licensing
Samples
Java
Android
LicensingSample
assets
bin
gen
libs
out
res
src
AndroidManifest.xml
LICENSE.txt
LicensingSample.iml
LicensingSample.ipr
LicensingSample.iws
lint.xml
project.properties
README.txt
Desktop
LabVIEW
Linux
OSX
Windows
Drag and drop a file into the space above to quickly locate it in the tree
Cancel

From the File menu select Project Structure to show the Project Structure settings.

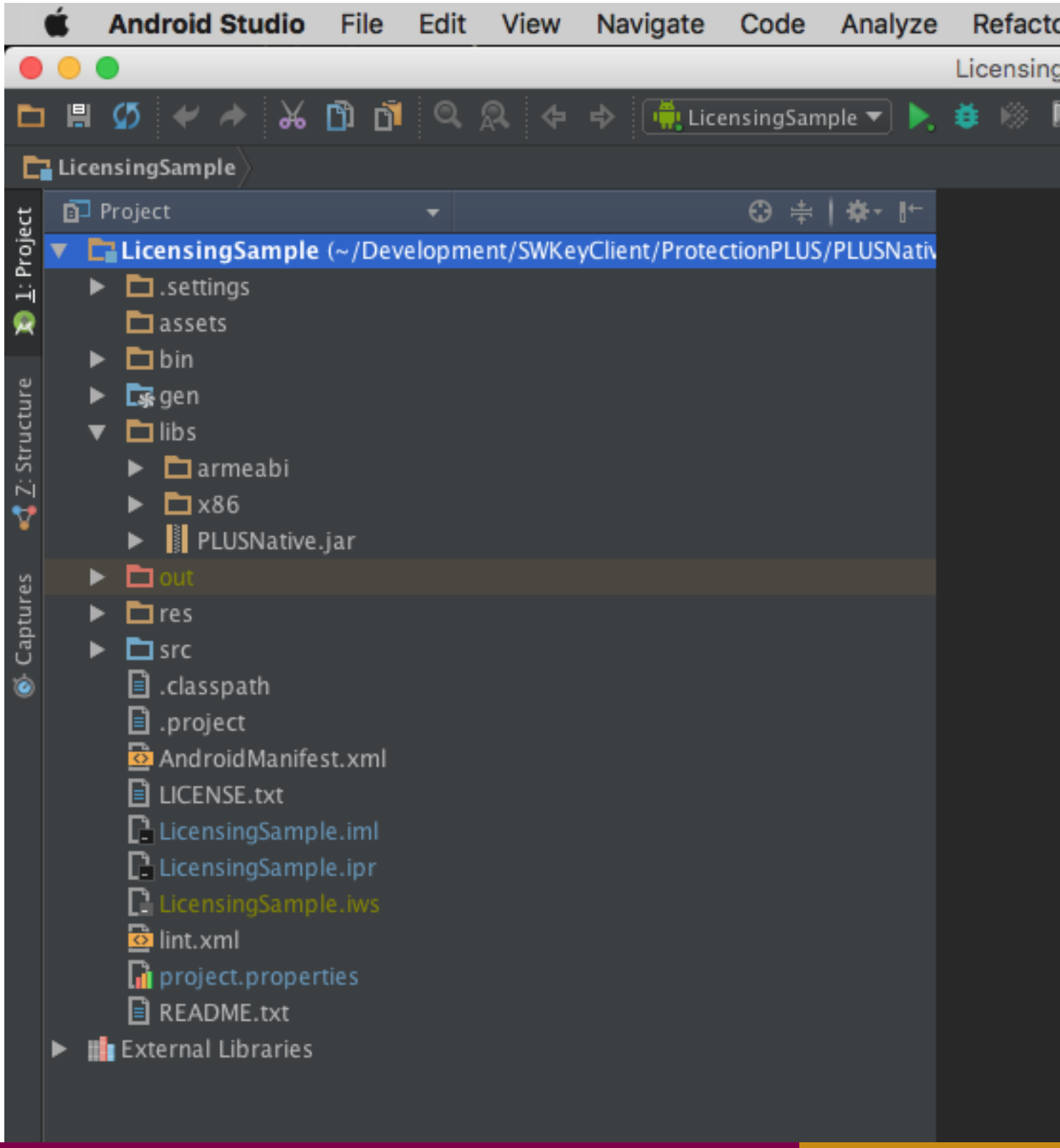


On the left click Project then on the right select your desired SDK in the drop-down and click the OK button.

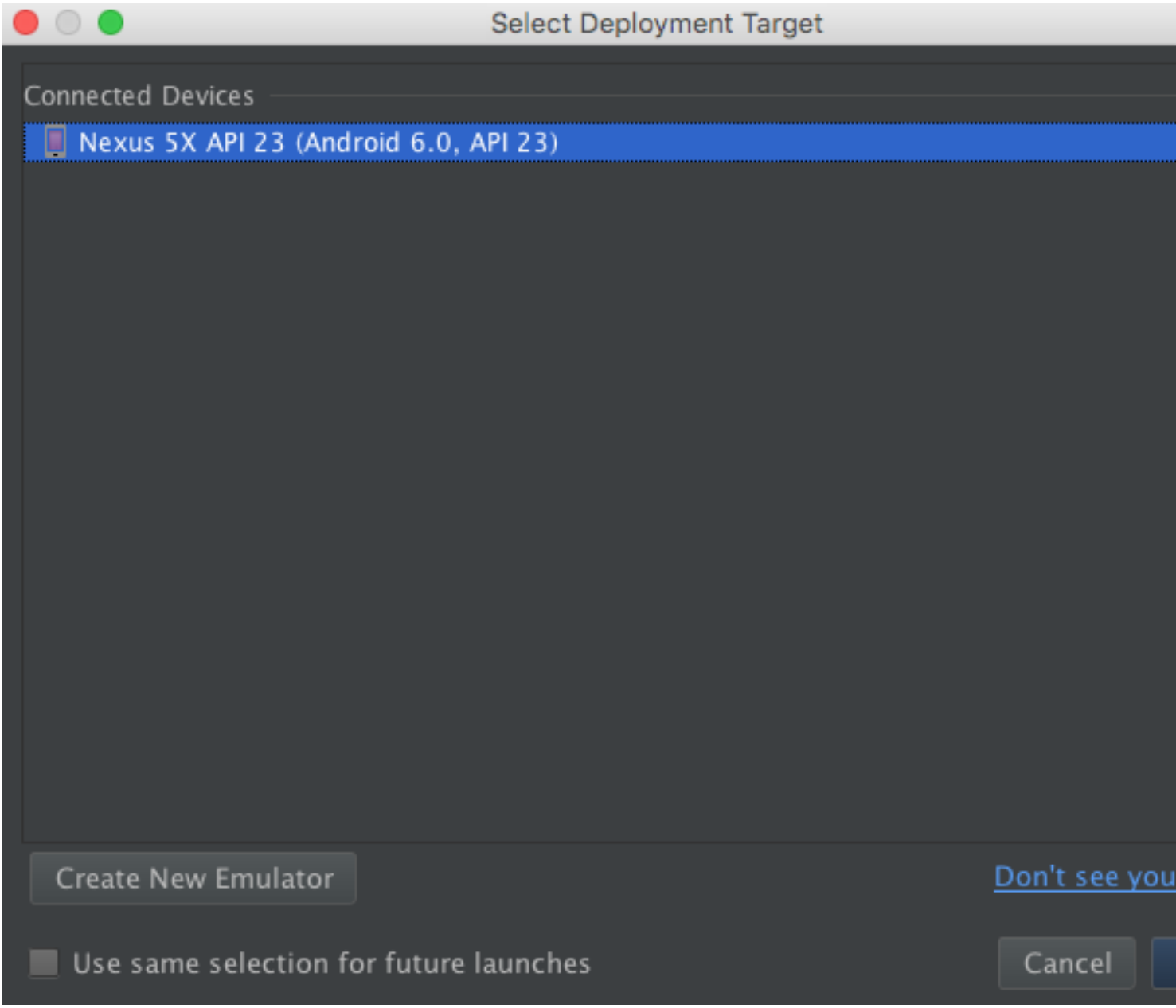


Connect your Android device. Optionally, you may use your preferred emulator.

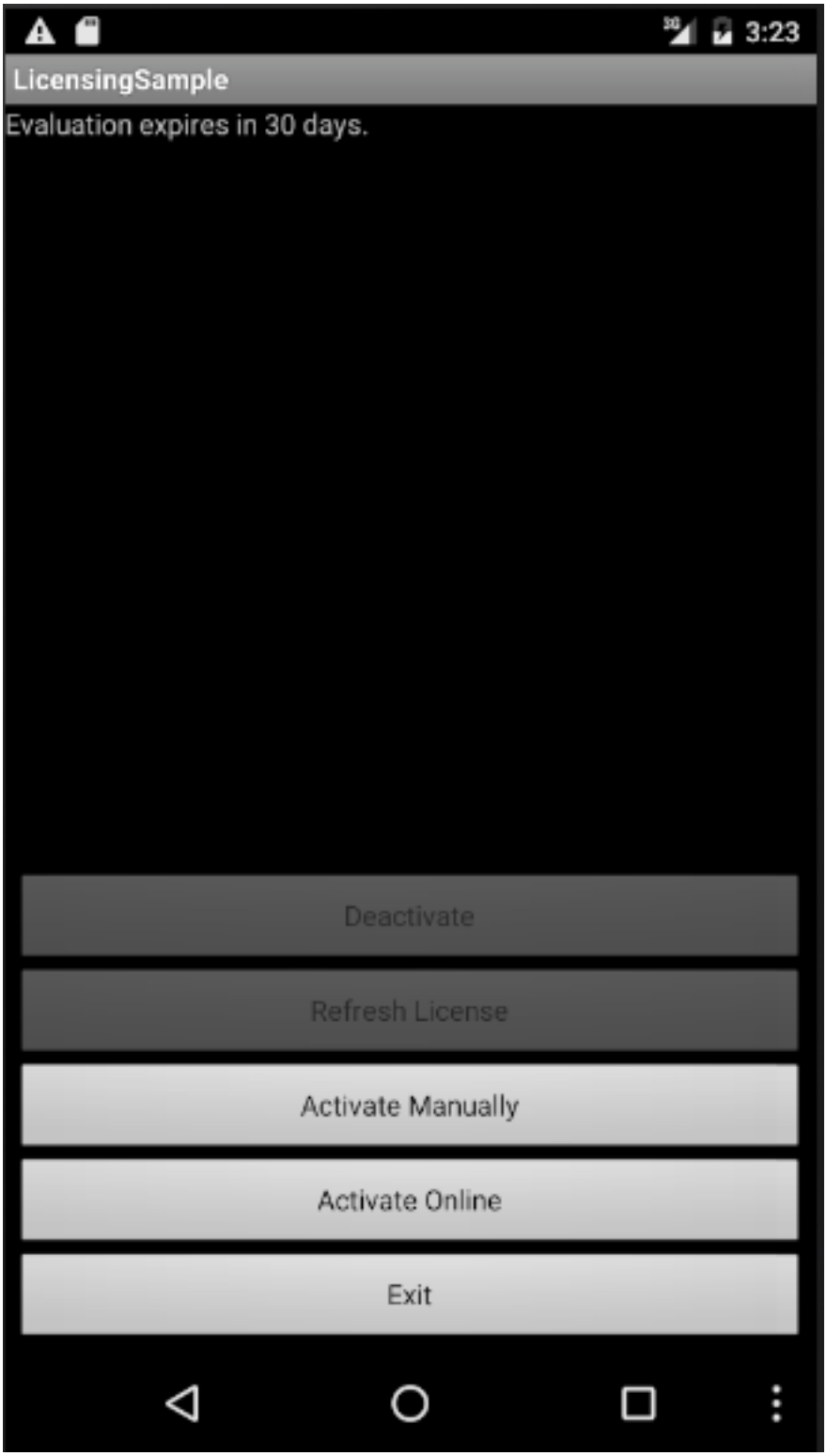
From the menu select Run/Run...


Android Studio File Edit View Navigate Code Analyze Refacto
Licensing
LicensingSample
LicensingSample
Project
LicensingSample (~/Development/SWKeyClient/ProtectionPLUS/PLUSNativ
.settings
assets
bin
gen
libs
armeabi
x86
PLUSNative.jar
out
res
src
.classpath
.project
AndroidManifest.xml
LICENSE.txt
LicensingSample.iml
LicensingSample.ipr
LicensingSample.iws
lint.xml
project.properties
README.txt
External Libraries
1: Project
7: Structure
Captures

Select the Device to use.

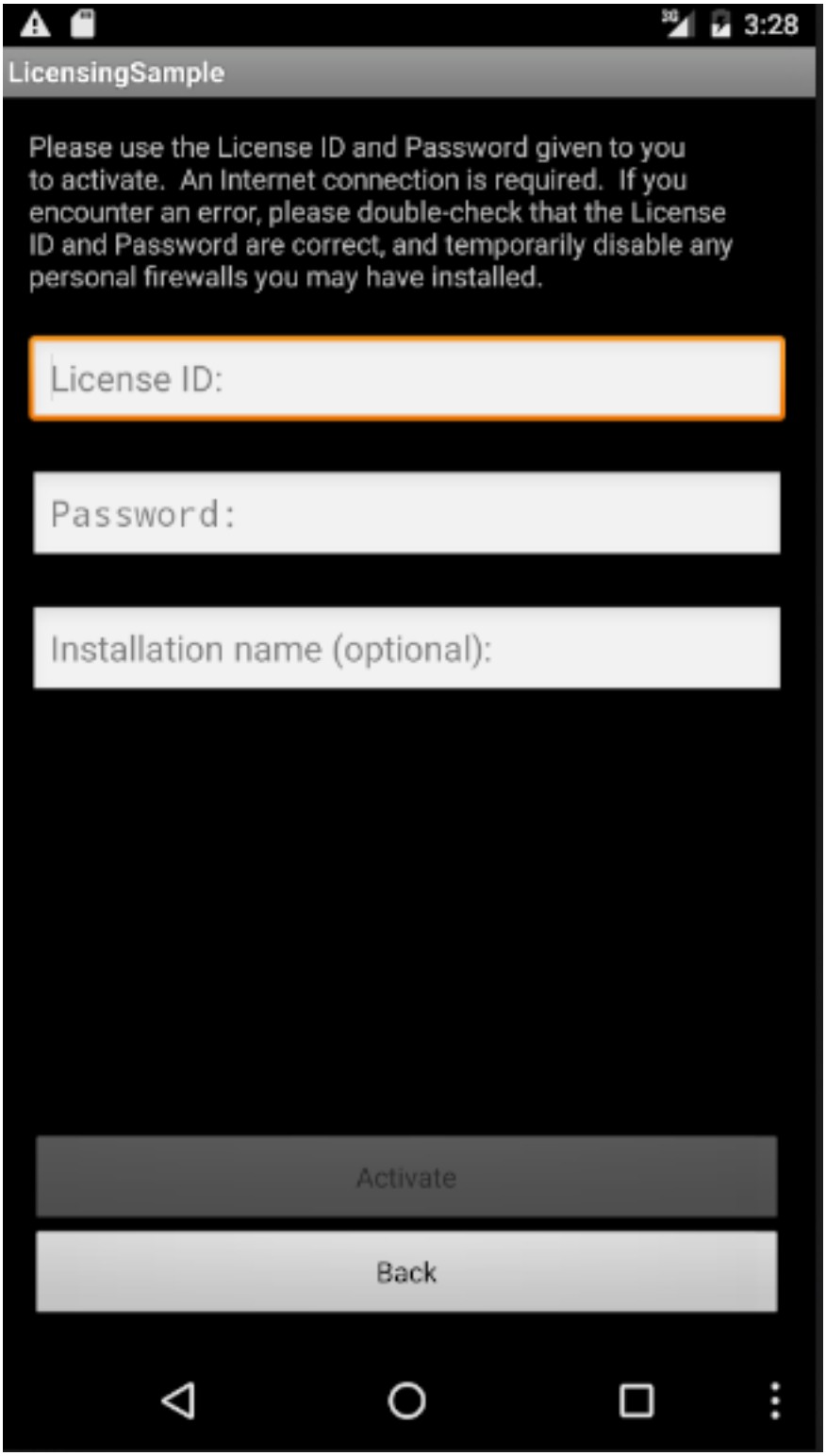


On the device you will see the main LicensingSample Activity.


3:23
LicensingSample
Evaluation expires in 30 days.
Deactivate
Refresh License
Activate Manually
Activate Online
Exit

Step 2 – Activating automatically with Instant SOLO Server

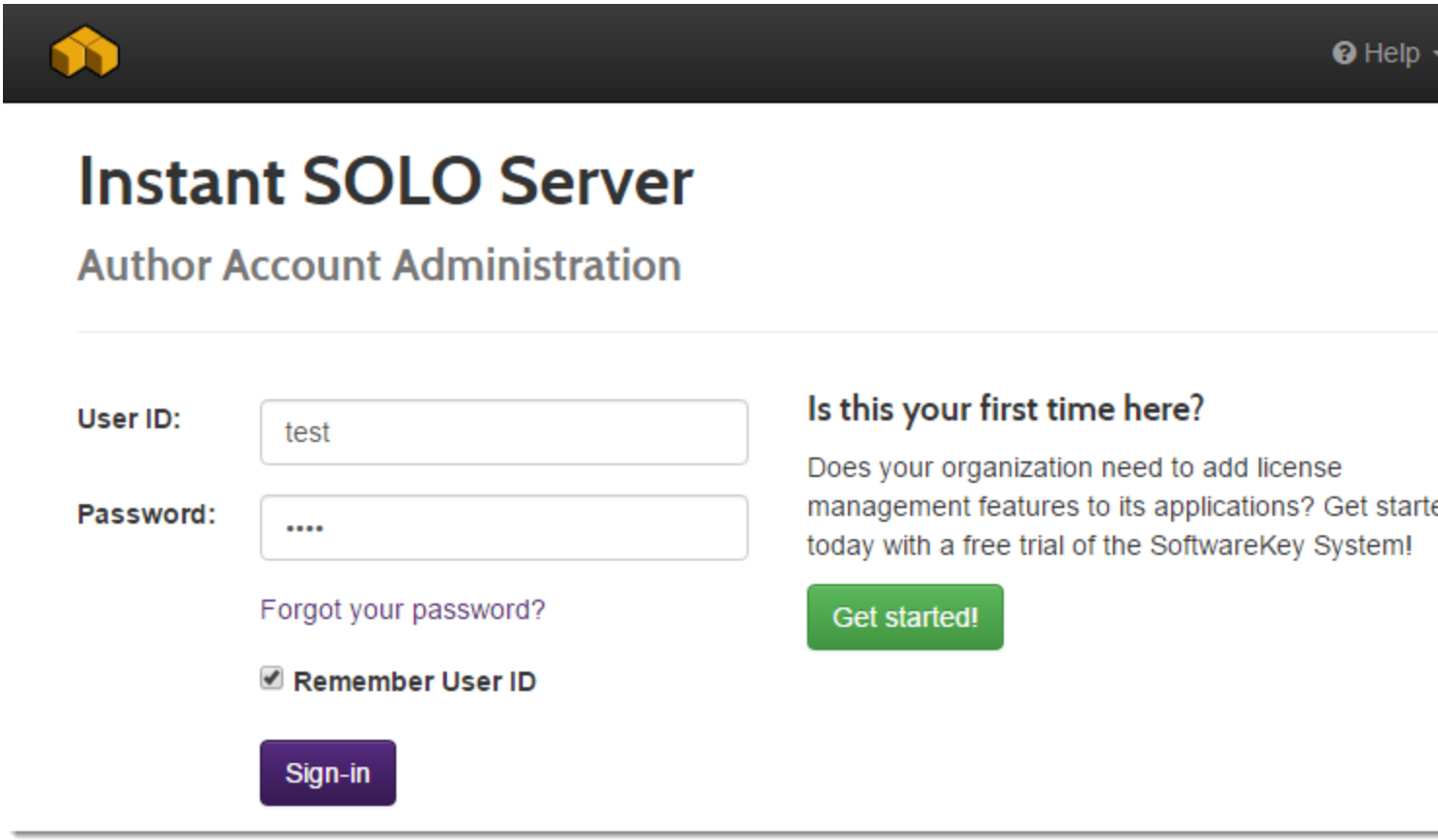
To activate the Sample Licensing Application automatically using SOLO Server, press the Activate Online button. A new Activity will prompt for a License ID and Password (both are required). Specifying an installation name is optional but helpful when performing multiple installations for different machines and/or users.


3:28
LicensingSample
Please use the License ID and Password given to you to activate. An Internet connection is required. If you encounter an error, please double-check that the License ID and Password are correct, and temporarily disable any personal firewalls you may have installed.
License ID:
Password:
Installation name (optional):
Activate
Back

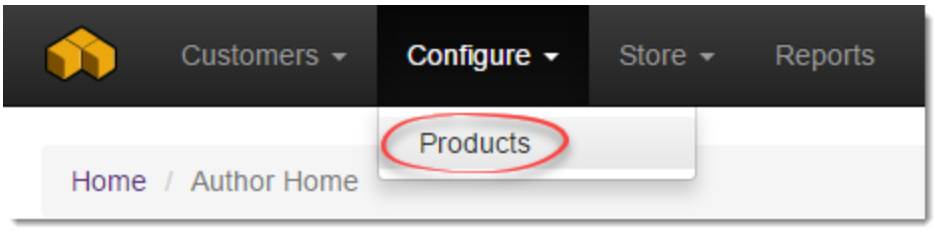
For ease of use, all of the Protection PLUS 5 SDK sample applications use a generic Test Author account with its own Encryption Key ID. We will use this Test Author account on Instant SOLO Server to generate the License ID and password necessary to activate the Licensing Sample

To **log into the Instant SOLO Server Test Account**, use the Test Author credentials given below:

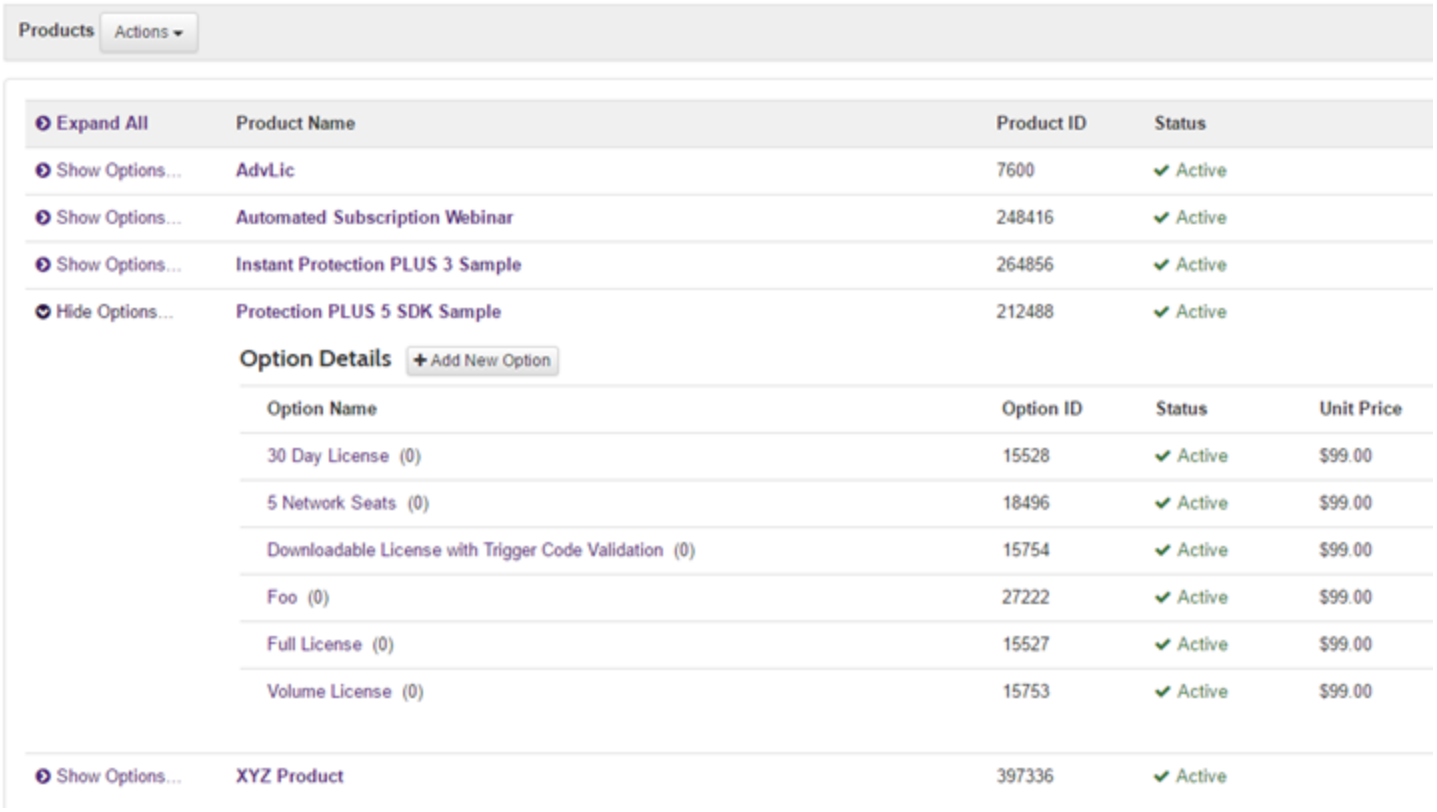
Login ID: test
Password: test



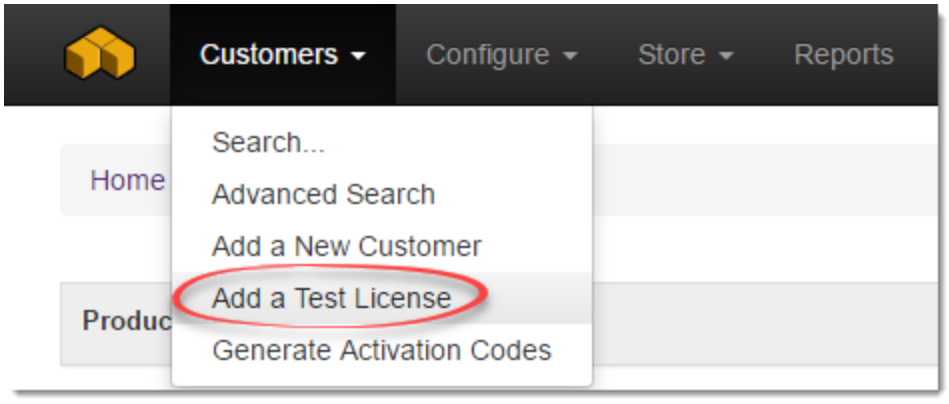
Once logged in, go to the menu *Configure / Products*.



The Product List page will show all of the products available to the Test Author. Expand 'Show Options' beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for activating the Licensing Sample application. For this tutorial, we will generate a license for the Full License.

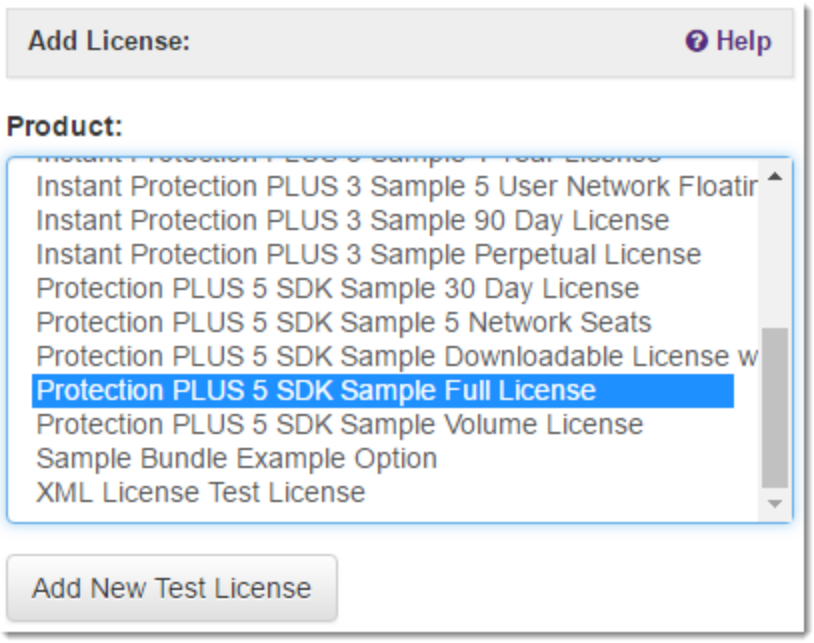


To add a test license for this product option, mouse over Customers in the navigation bar, and click 'Add Test License'.



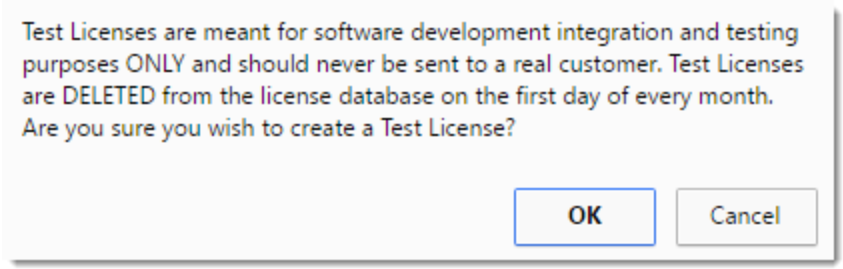
>

Select the Protection PLUS 5 SDK Sample Full License and click Add New Test License.

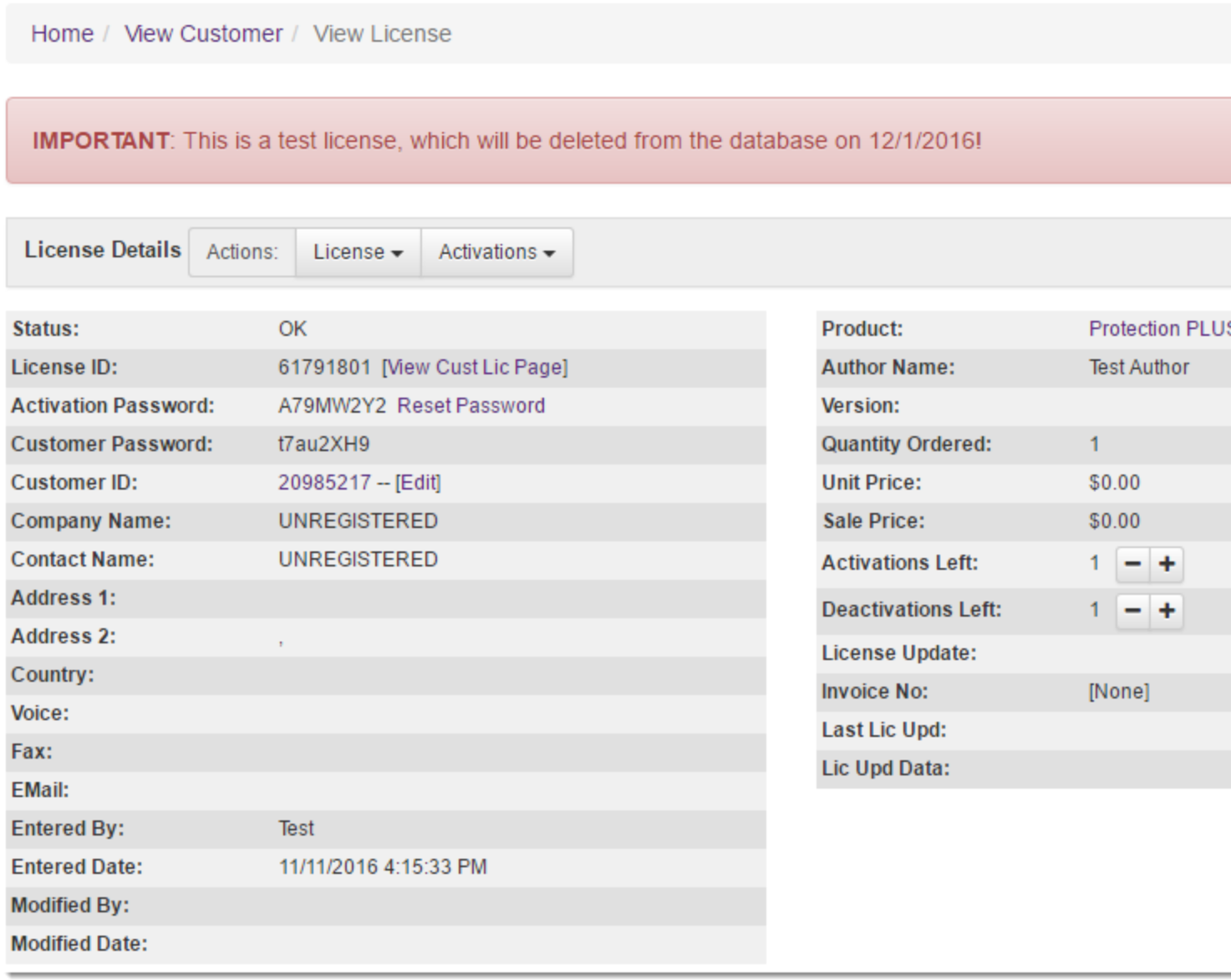


Important:

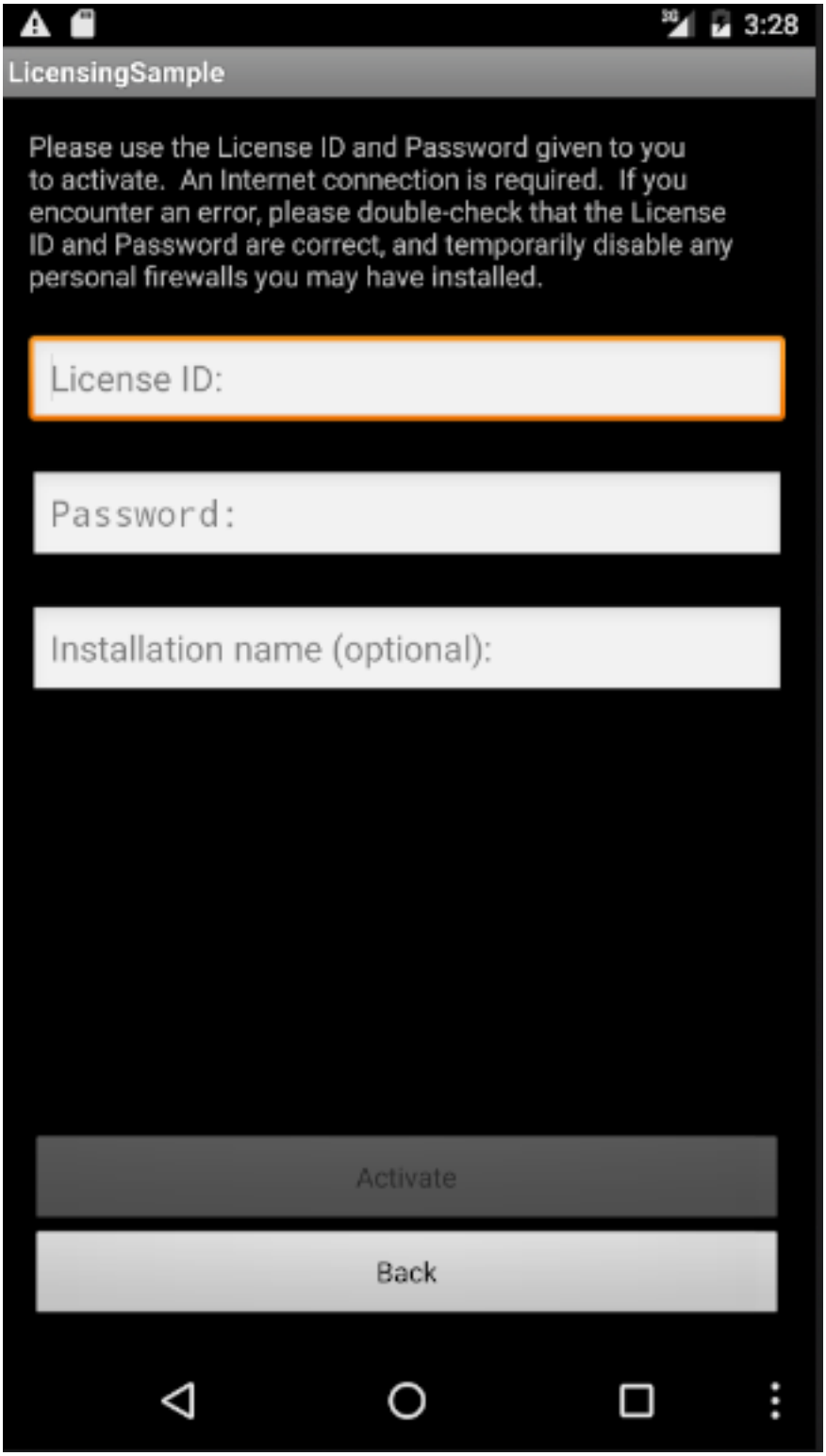
Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.



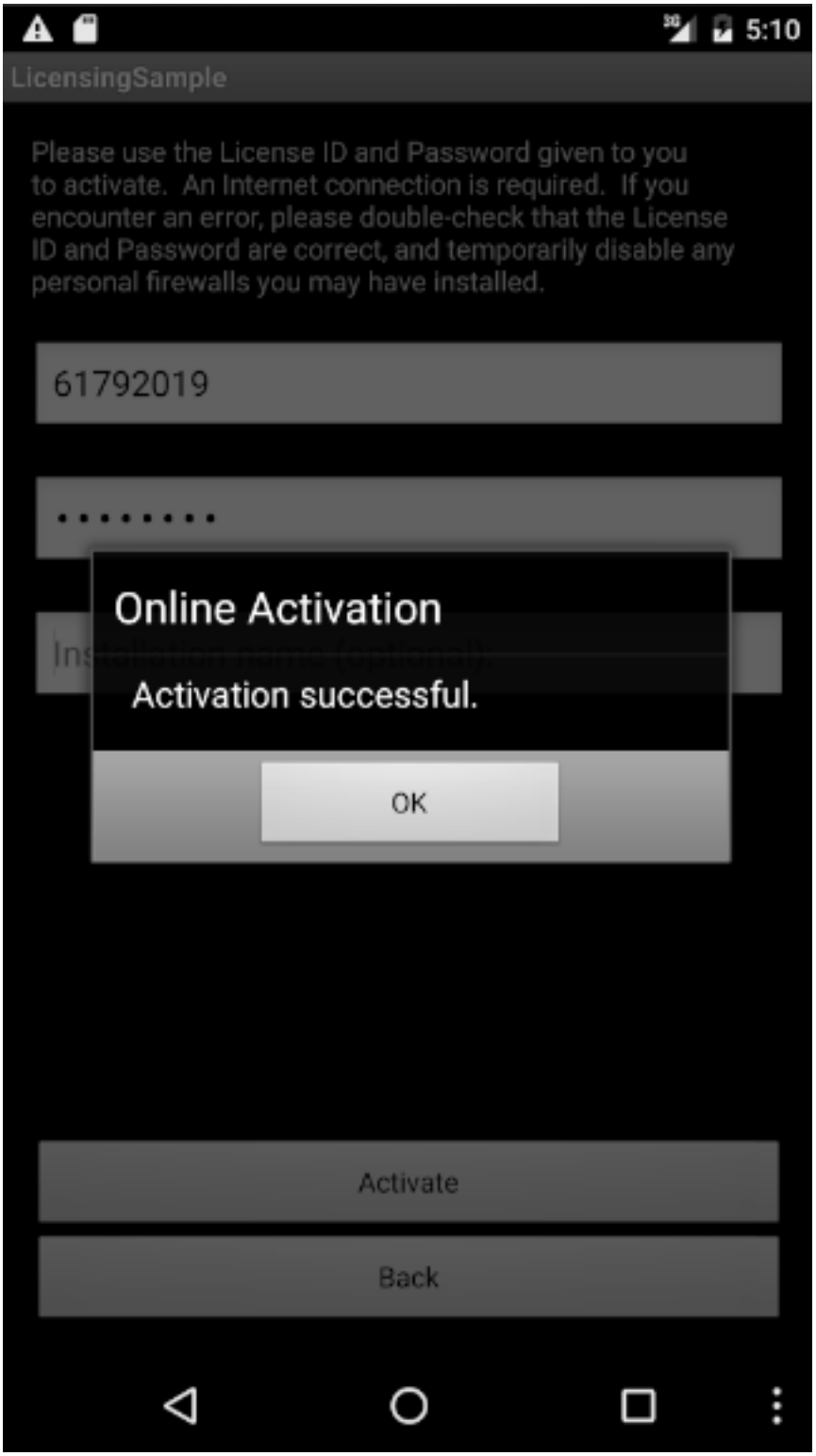
The Test License has now been created. We now have a License ID and Password to continue testing the Licensing Sample.



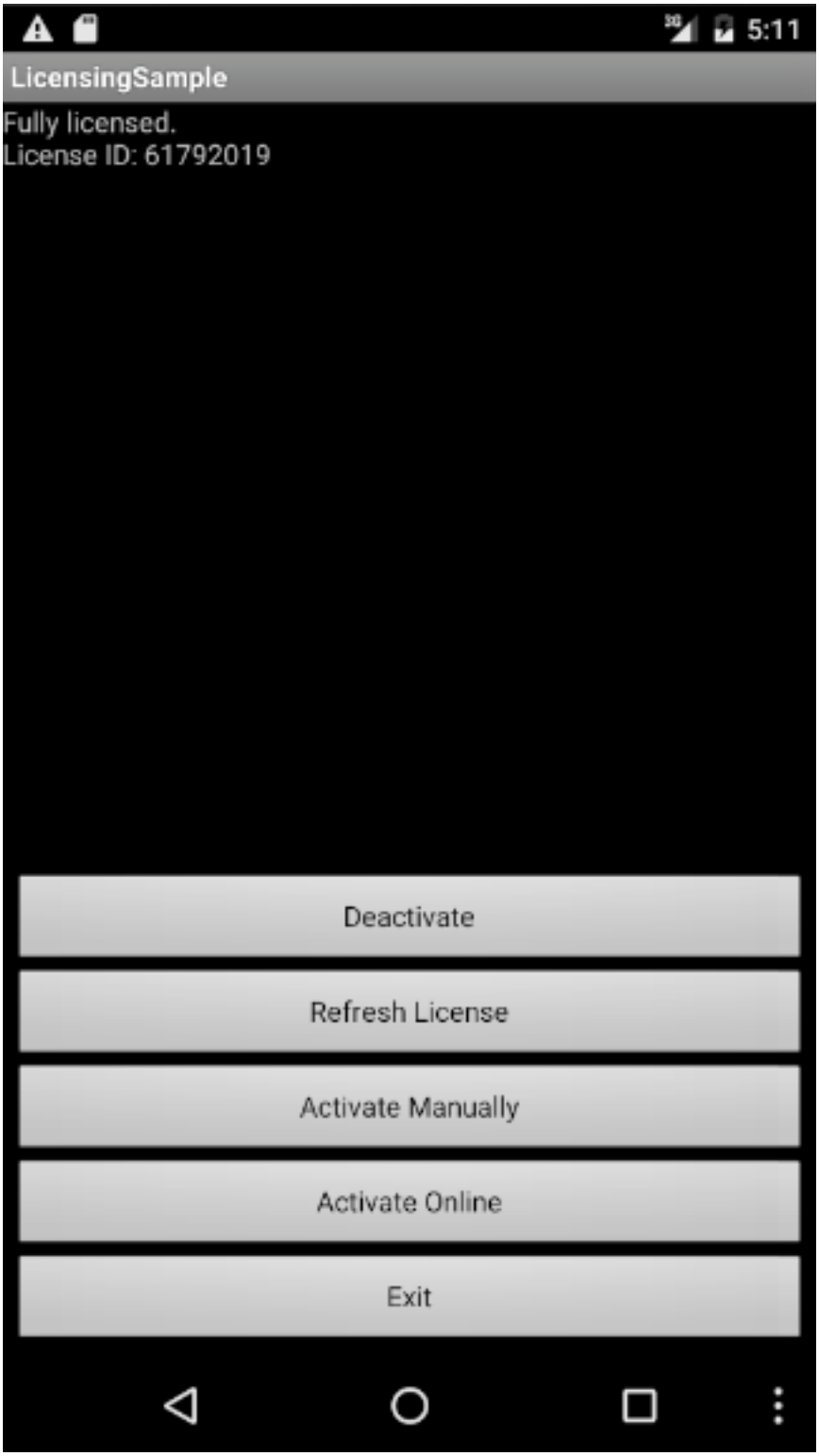
Return to the activation prompt for our Licensing Sample. Enter the License ID and Password in their respective fields and, optionally, an Installation Name. Press Activate to communicate with Instant SOLO Server for license validation.


3:28
LicensingSample
Please use the License ID and Password given to you to activate. An Internet connection is required. If you encounter an error, please double-check that the License ID and Password are correct, and temporarily disable any personal firewalls you may have installed.
License ID:
Password:
Installation name (optional):
Activate
Back

Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.

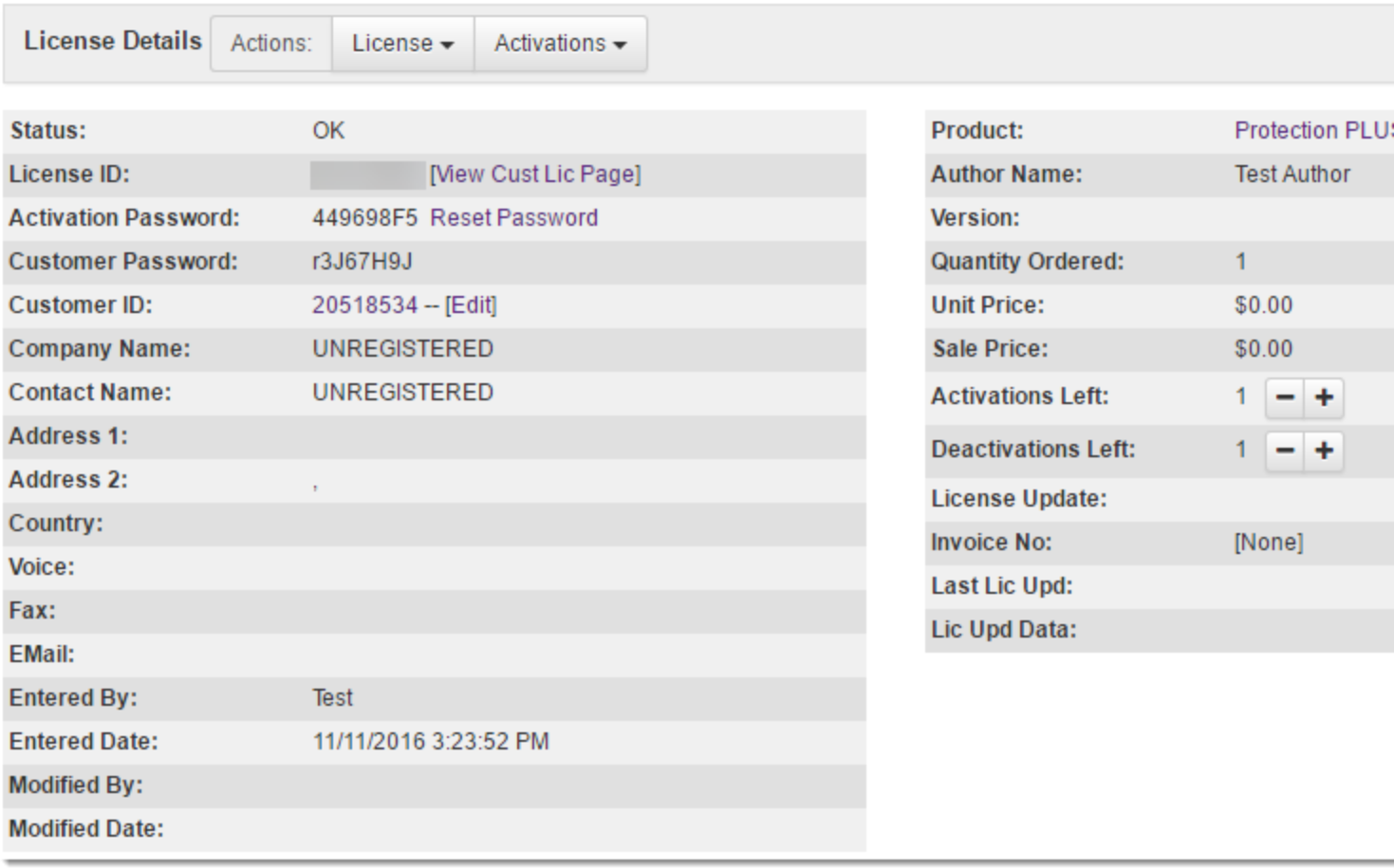

5:10
LicensingSample
Please use the License ID and Password given to you to activate. An Internet connection is required. If you encounter an error, please double-check that the License ID and Password are correct, and temporarily disable any personal firewalls you may have installed.
61792019
Online Activation
Activation successful.
OK
Activate
Back

The License Status will now show as Fully Licensed.

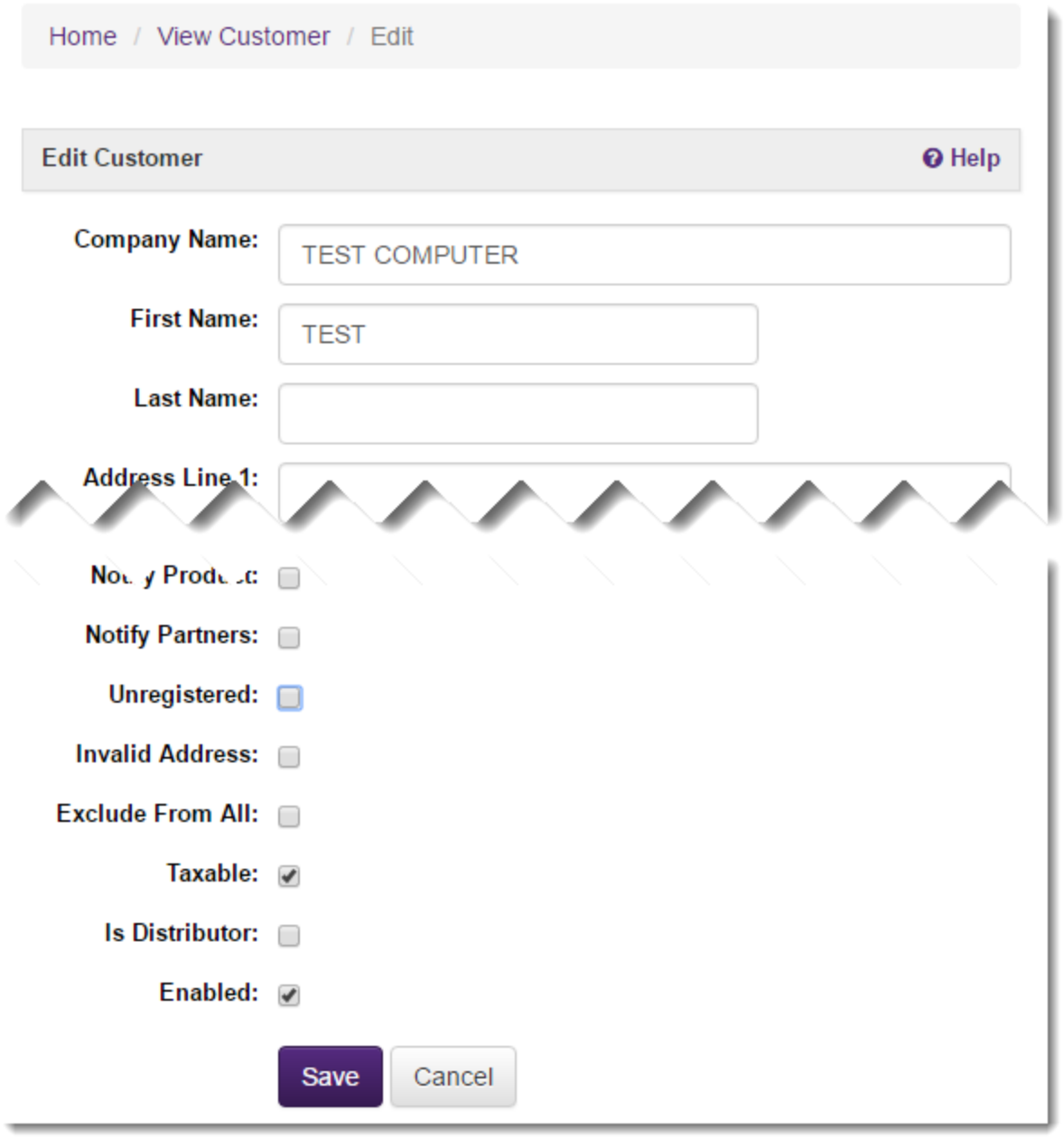

5:11
LicensingSample
Fully licensed.
License ID: 61792019
Deactivate
Refresh License
Activate Manually
Activate Online
Exit

Step 3 - Refresh a License

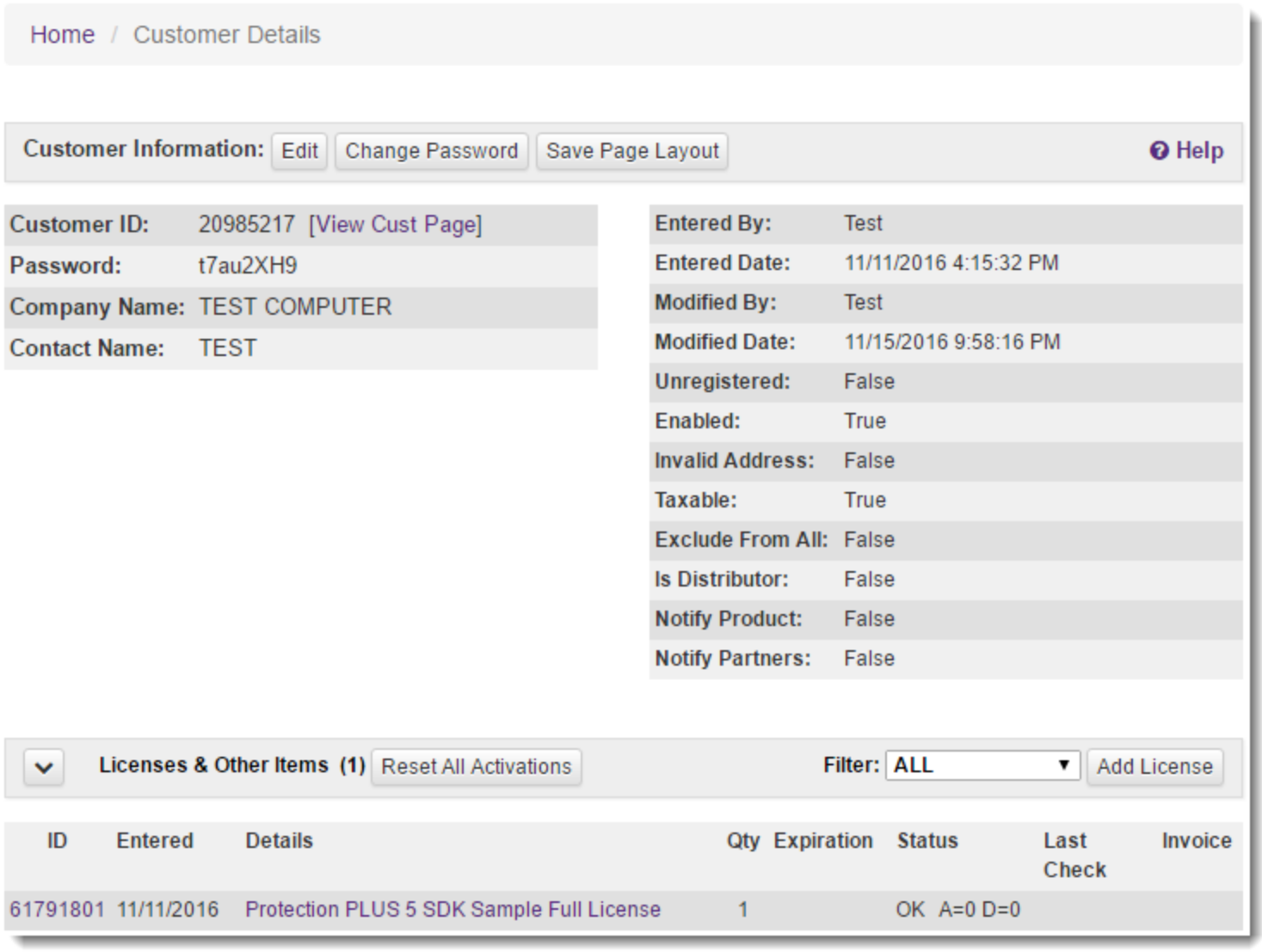
A license can be refreshed with updated information sent from Instant SOLO Server. We can modify customer details from the License Details page on Instant SOLO Server. On the License Details page, click Edit, located to the right of the Customer ID.



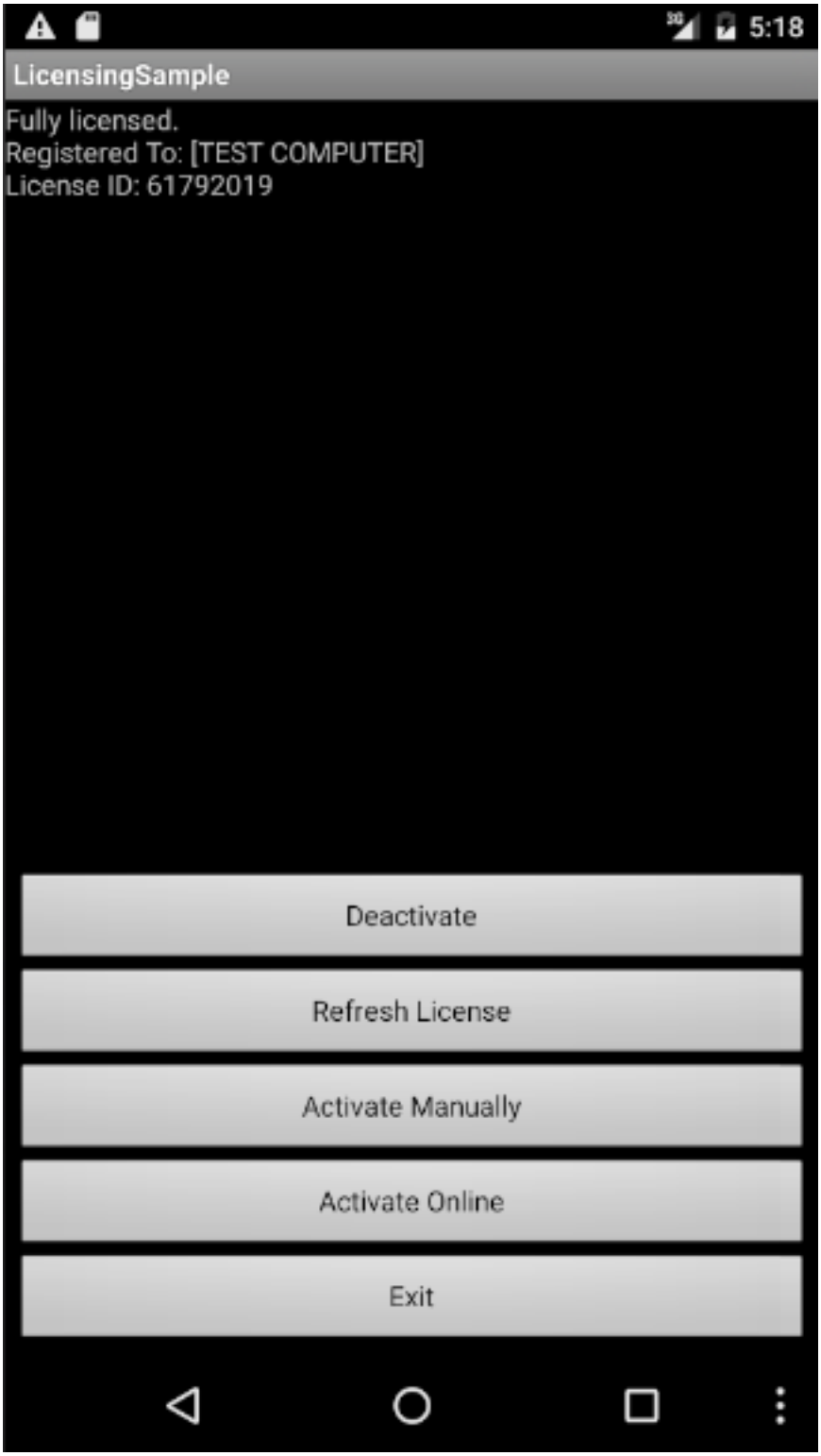
Modify the Company Name and additional fields as you please. Uncheck the check box setting to the right of Unregistered. Scroll to the bottom and click Save.



After the Customer Details have been saved, the page returns to the Customer Information page. This page includes customer details we have edited in addition to licenses and previous orders.

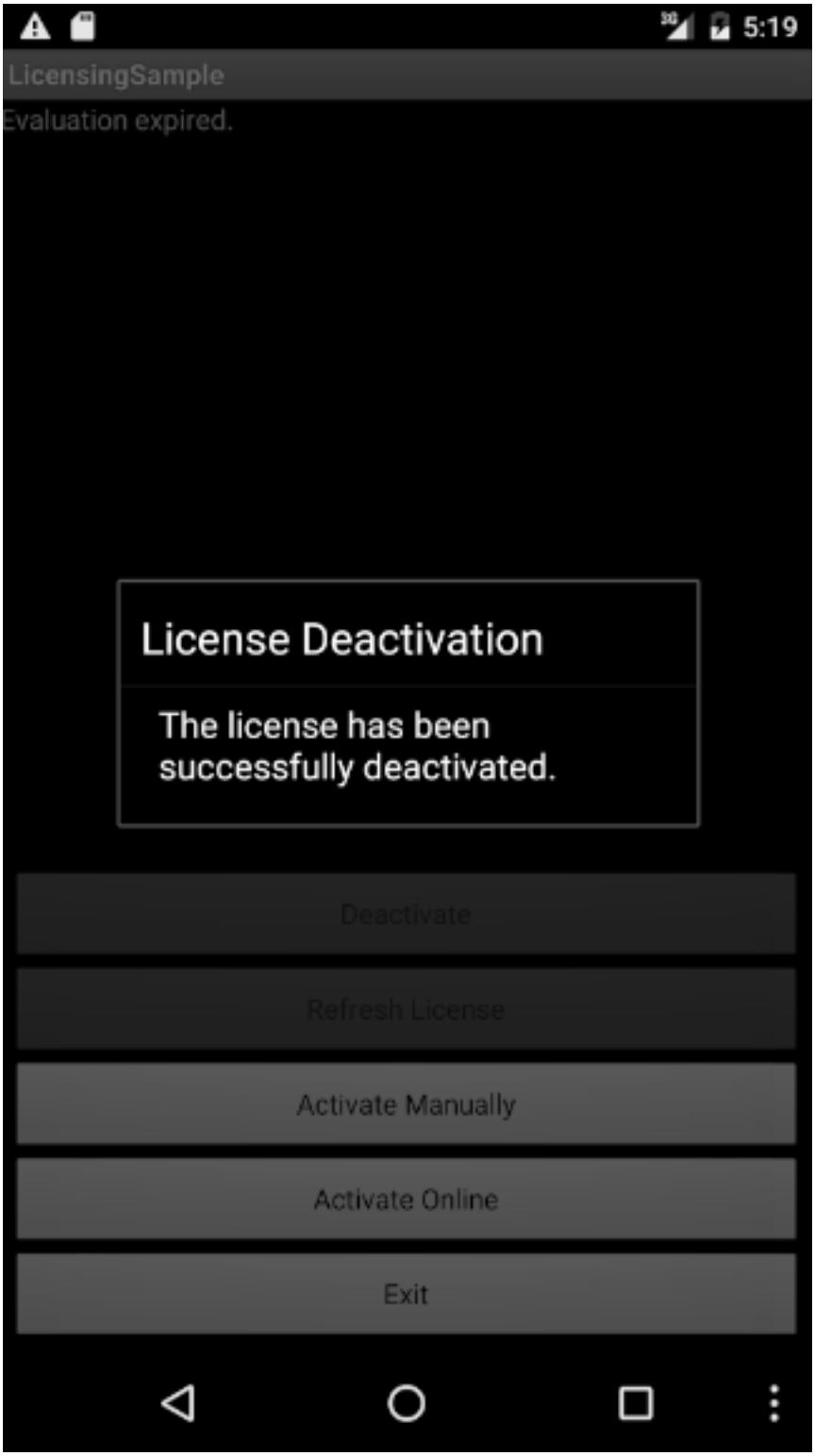


Return to the Licensing Sample Main Activity to proceed with a license refresh. Press Refresh License. If the license successfully refreshed, the License Status will now update with the modified customer details.

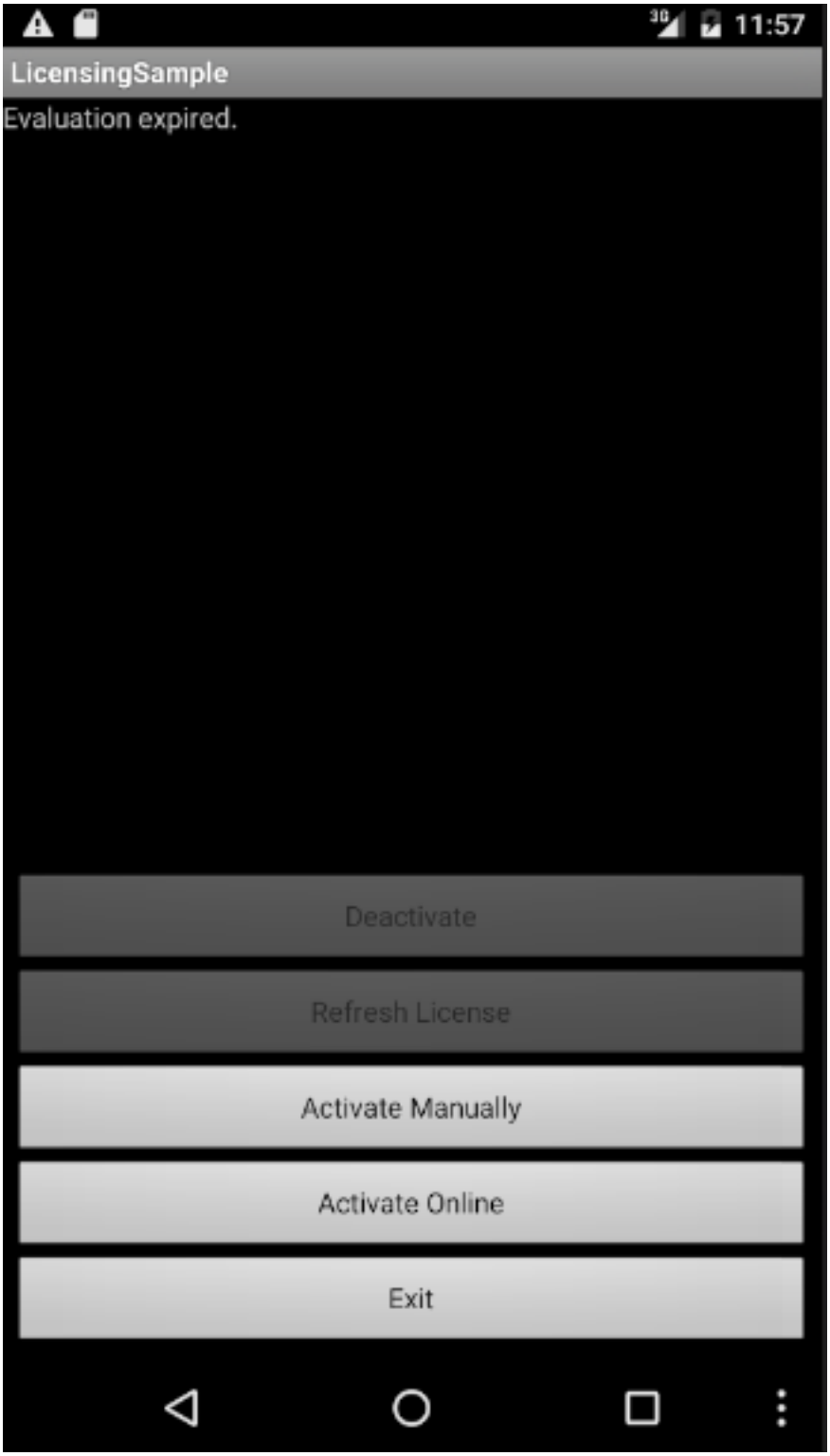

5:18
LicensingSample
Fully licensed.
Registered To: [TEST COMPUTER]
License ID: 61792019
Deactivate
Refresh License
Activate Manually
Activate Online
Exit

Step 4 - Process a Manual Activation

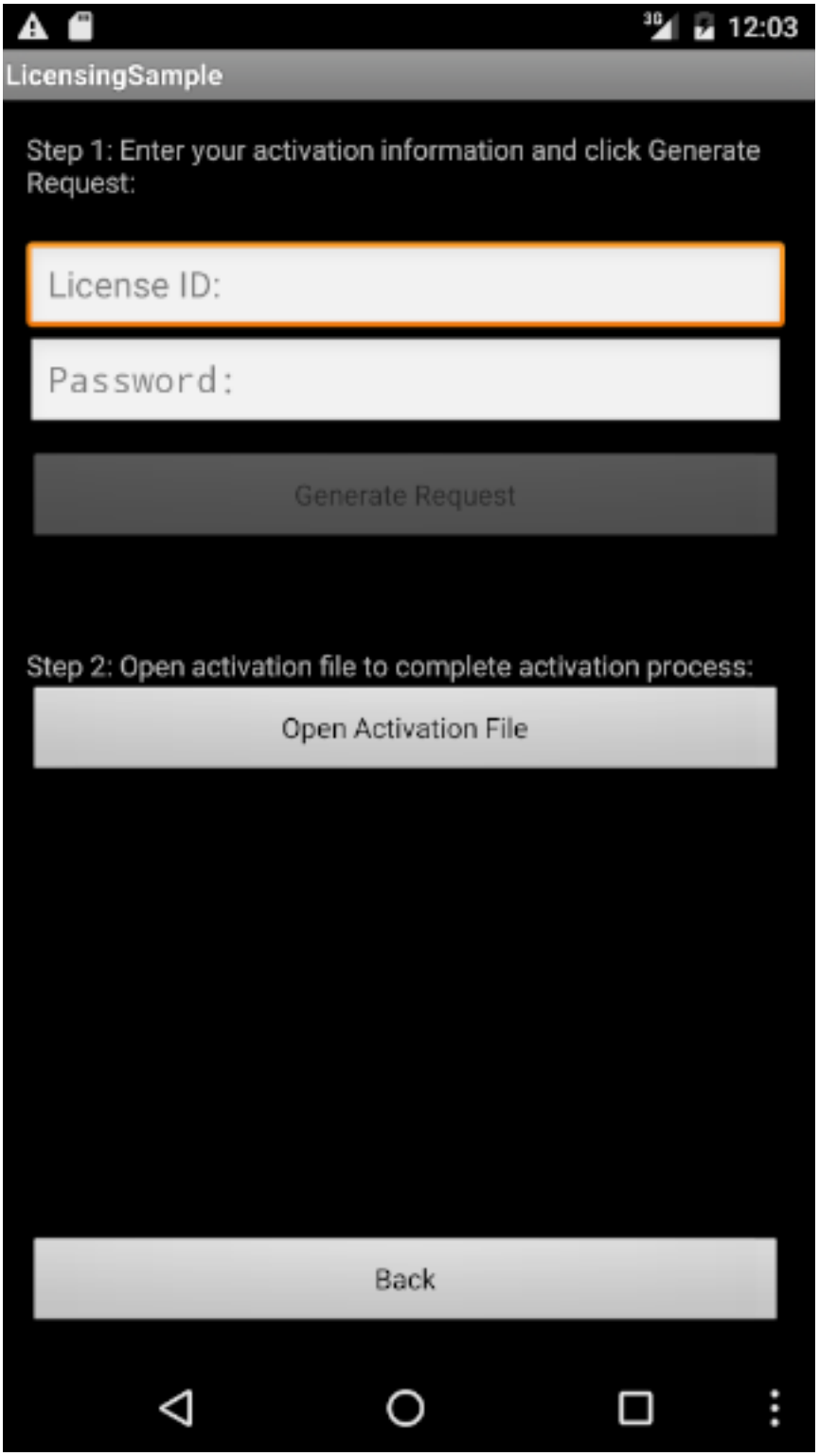
If you previously activated the Licensing Sample online using Instant SOLO Server, you will need to deactivate the license before testing manual activations. In the Main Activity of the Licensing Sample press Deactivate. A confirmation will appear if the license was successfully deactivated.


5:19
LicensingSample
Evaluation expired.
License Deactivation
The license has been successfully deactivated.
Deactivate
Refresh License
Activate Manually
Activate Online
Exit

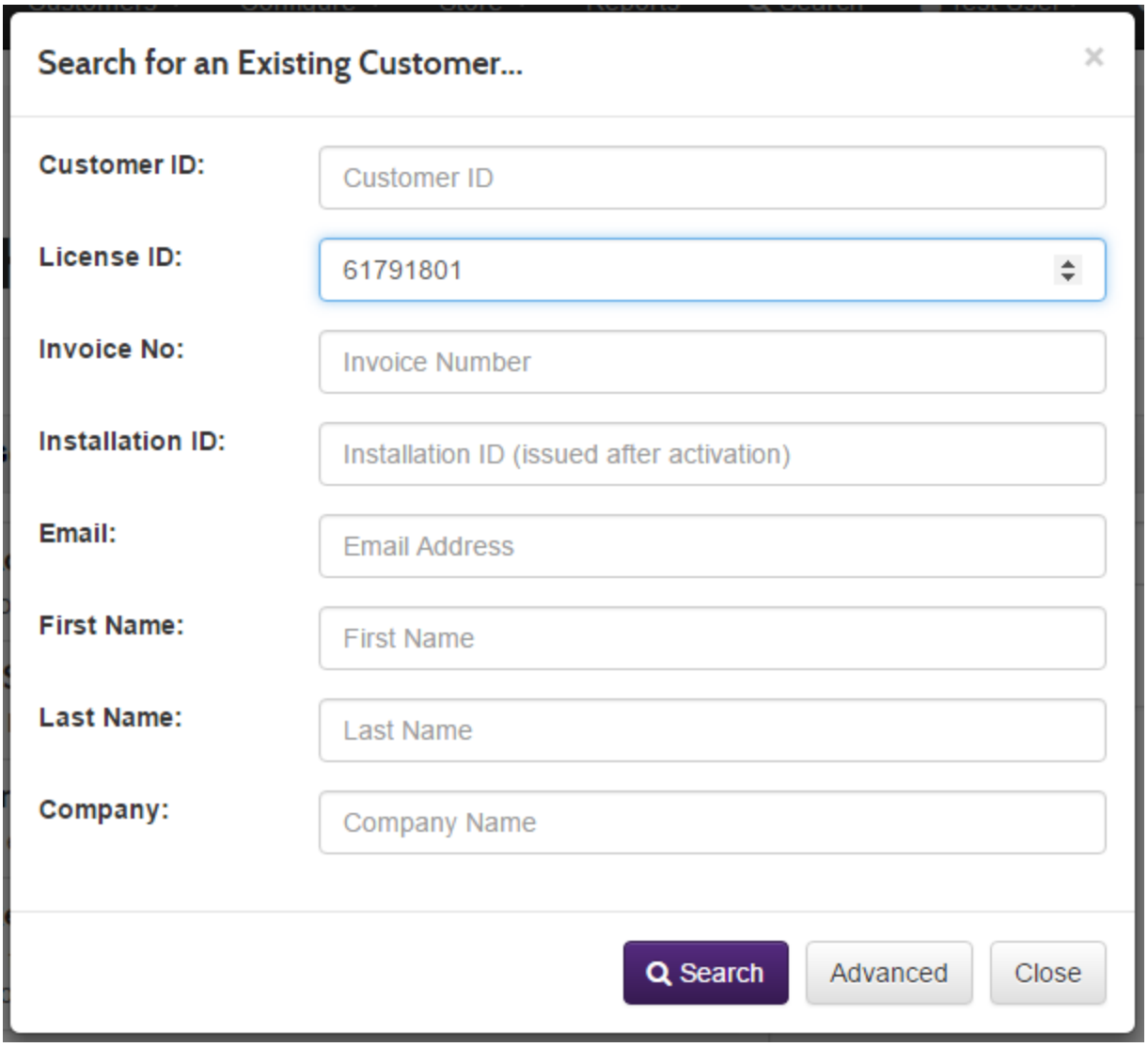
The License Status will now show "Evaluation expired."


11:57
LicensingSample
Evaluation expired.
Deactivate
Refresh License
Activate Manually
Activate Online
Exit

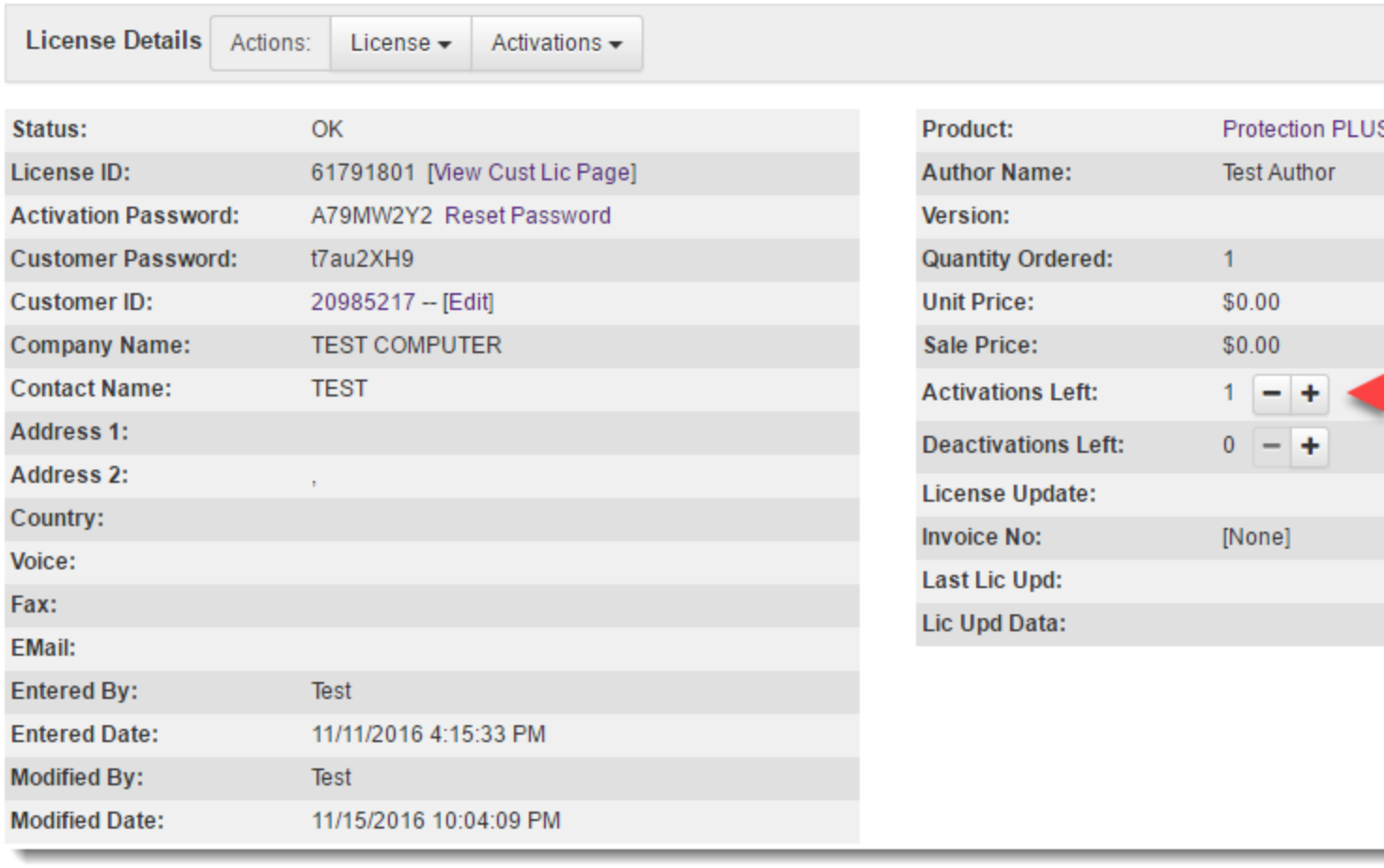
Press Activate Manually to proceed with testing a manual activation. A manual activation allows for activations from another computer or by e-mail.


12:03
LicensingSample
Step 1: Enter your activation information and click Generate Request:
License ID:
Password:
Generate Request
Step 2: Open activation file to complete activation process:
Open Activation File
Back

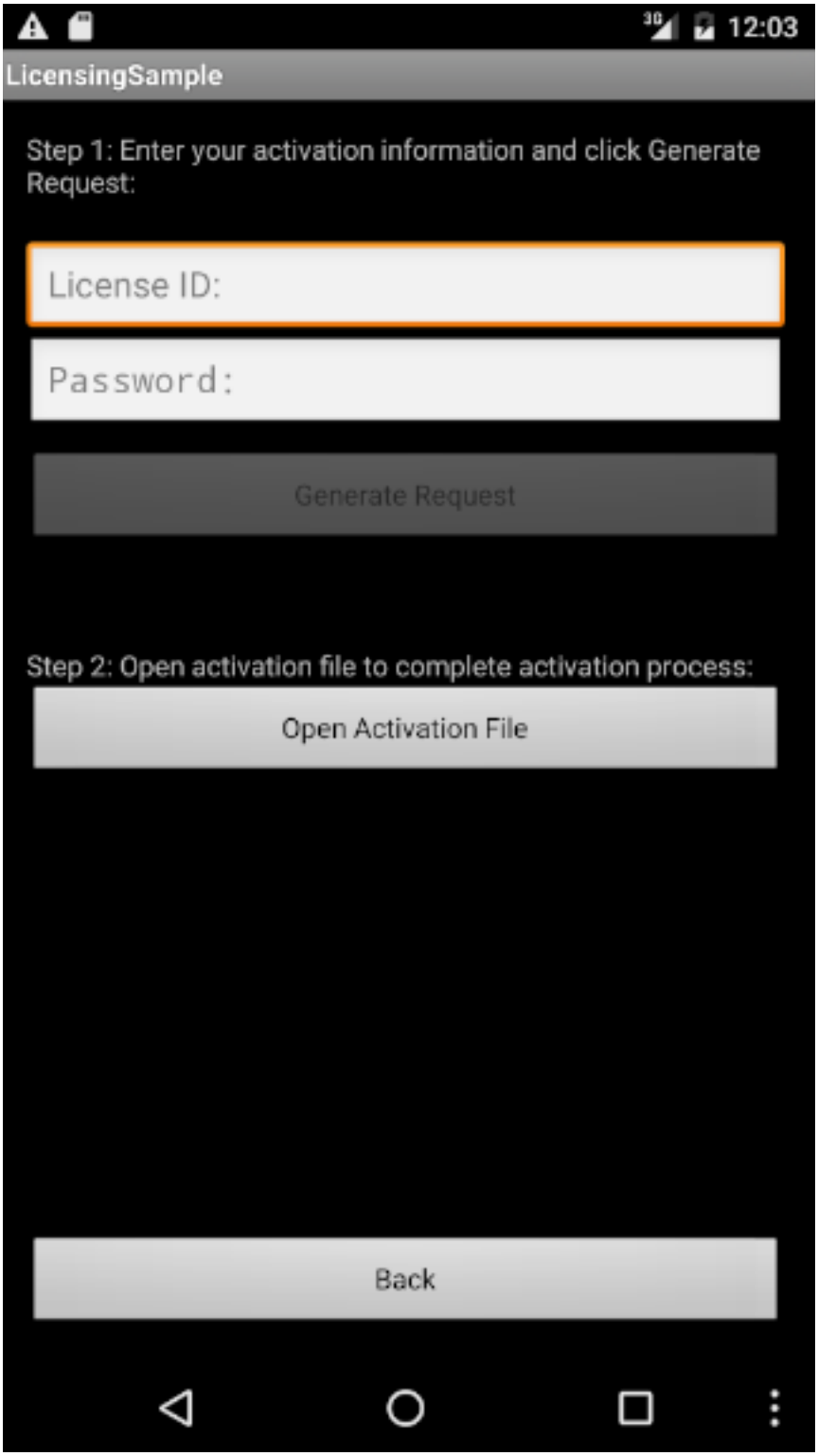
Return to Instant SOLO Server using the Test Author account. Perform a quick search for the License ID previously used to activate online (If you no longer have this number, proceed to create a new License ID as shown in Step # 2).



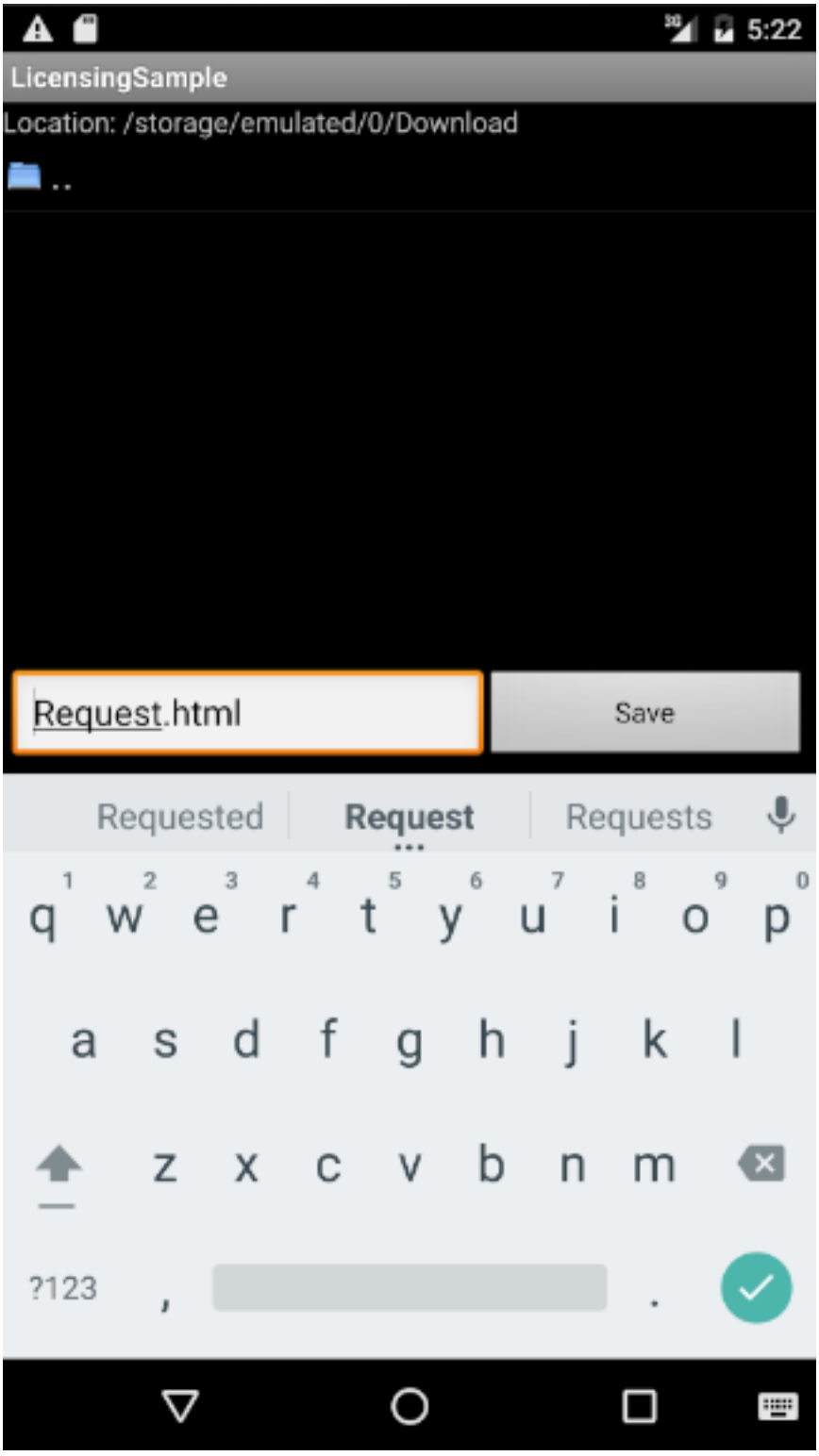
When we clicked on deactivate license, Instant SOLO Server automatically incremented the amount of activations left by 1. This setting was predefined in the Product Option ID we selected for testing. If you need to manually increase the number of activations left, you can increment this number by clicking the + button to the right of the Activations Left field.



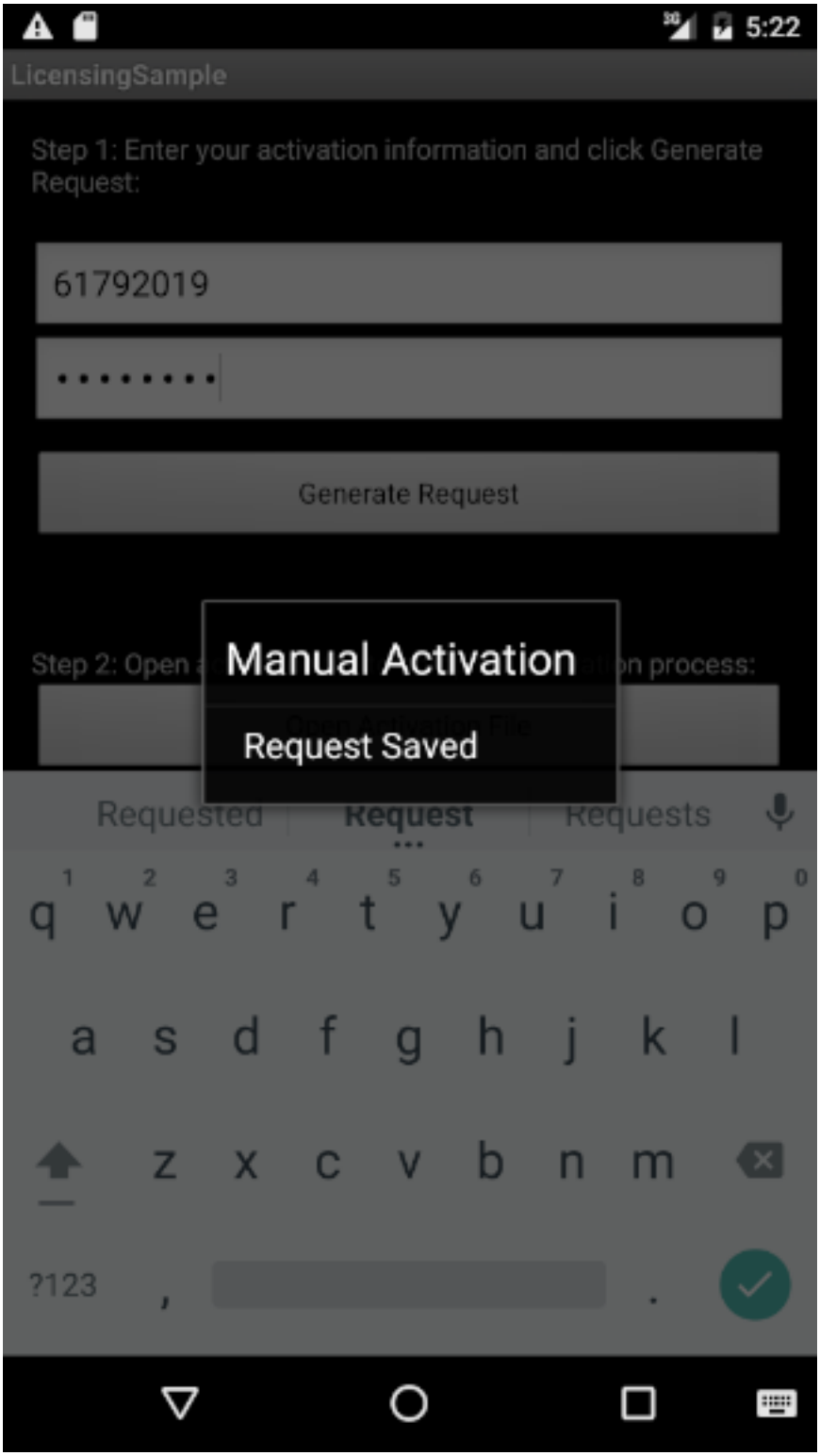
Return to the Activate Manually screen for our Licensing Sample. Enter the License ID and Password in their respective fields and press Generate Request as it will then be enabled.


12:03
LicensingSample
Step 1: Enter your activation information and click Generate Request:
License ID:
Password:
Generate Request
Step 2: Open activation file to complete activation process:
Open Activation File
Back

Browse to a location to save the file.

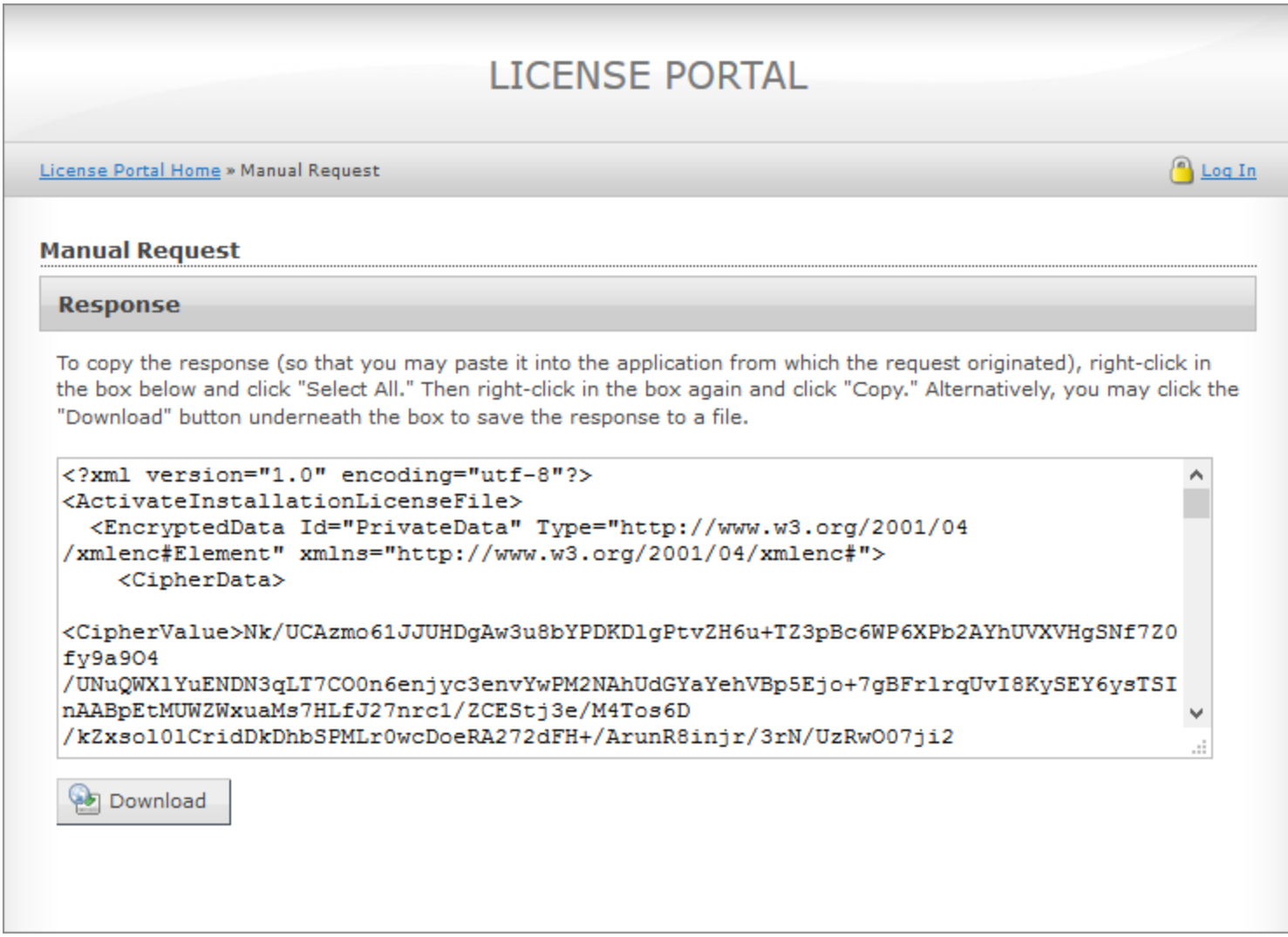

5:22
LicensingSample
Location: /storage/emulated/0/Download
..
Request.html
Save
Requested
Request
Requests
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123
,
.

Save the generated request file.

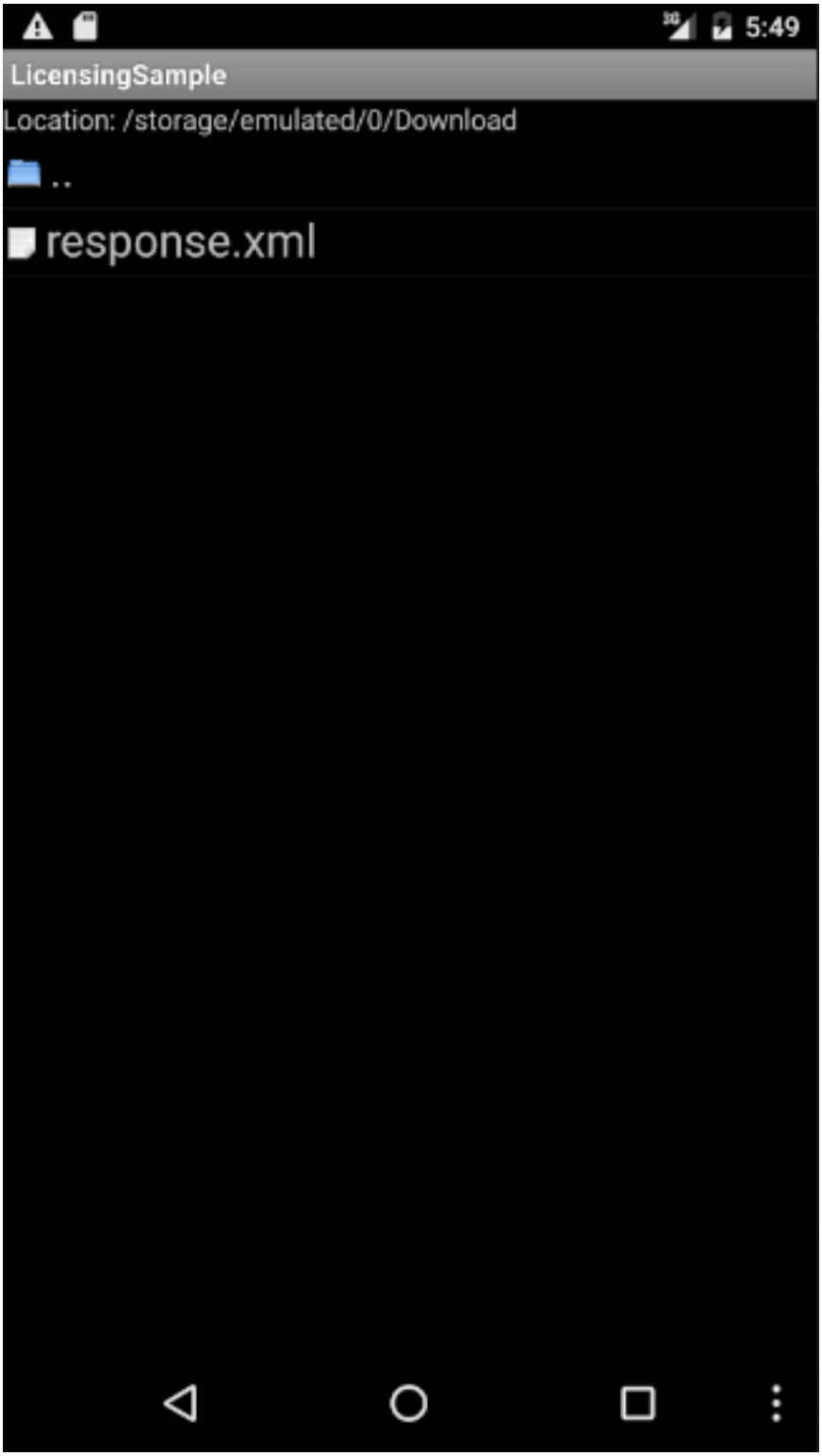

5:22
LicensingSample
Step 1: Enter your activation information and click Generate Request:
61792019
Generate Request
Manual Activation
Request Saved
Step 2: Open
process:
Requested
Request
Requests

Transfer this request file to your computer and double-click it. Your computer's browser will open to the License Portal page.

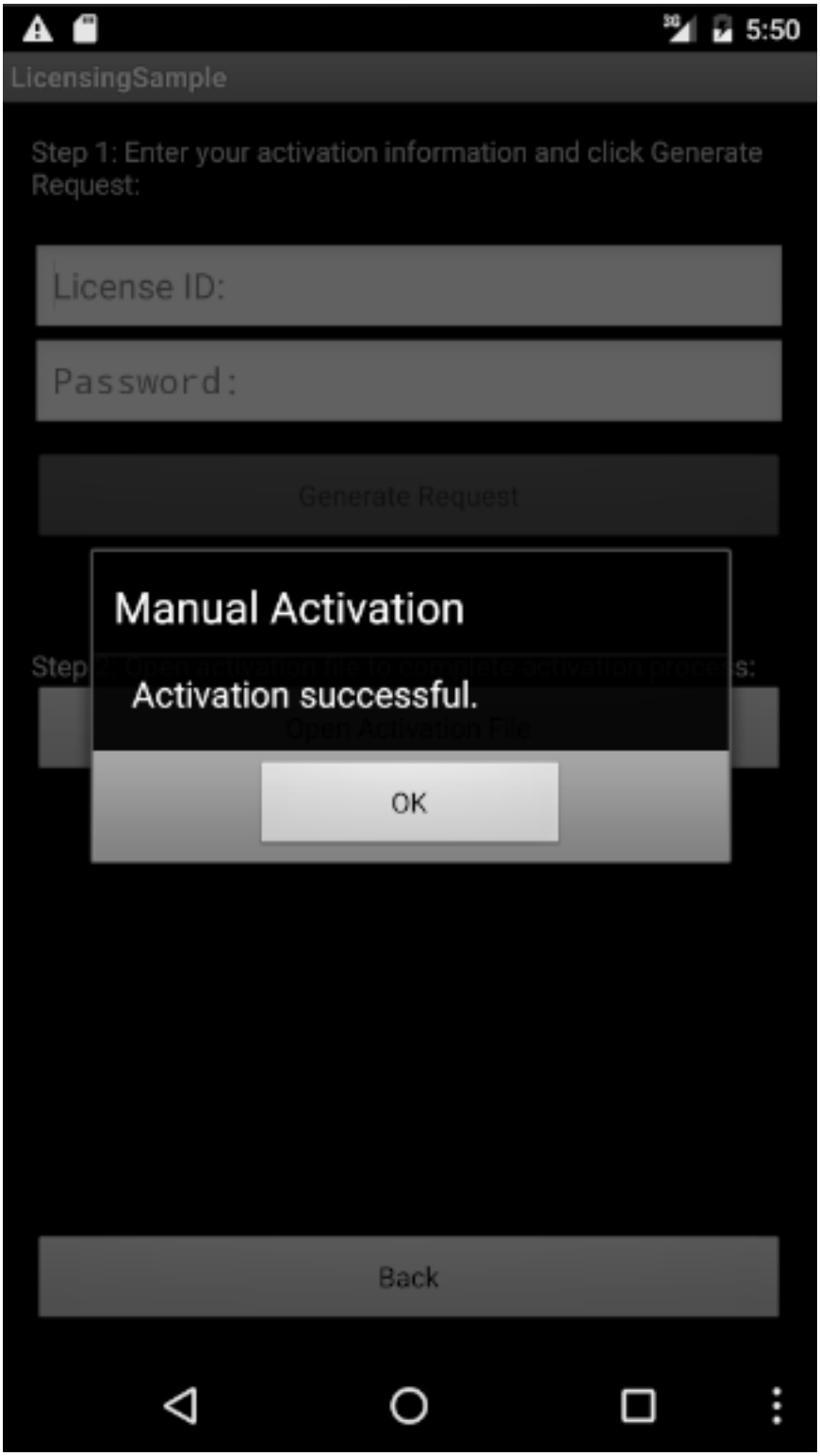
The License Portal will now generate an encrypted response. Click the Download button and save the response file to the computer. Transfer this file back to the Android device.



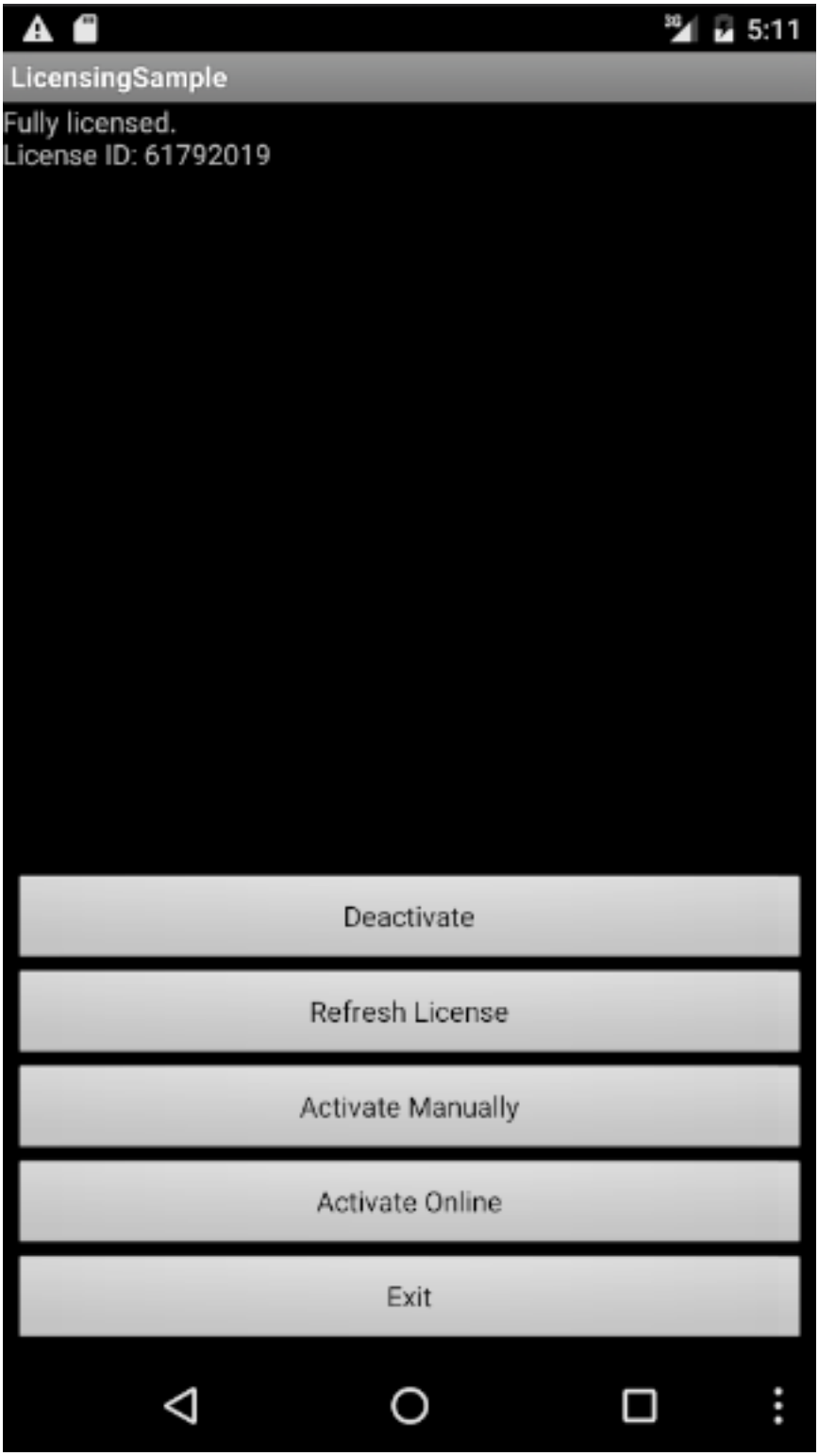
Return to the activation and press the Open Activation File button. Browse to the response file you transferred back to your device.


5:49
LicensingSample
Location: /storage/emulated/0/Download
..
response.xml

Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.


5:50
LicensingSample
Step 1: Enter your activation information and click Generate Request:
License ID:
Password:
Generate Request
Manual Activation
Activation successful.
OK
Back

The License Status will now show as Fully Licensed.


5:11
LicensingSample
Fully licensed.
License ID: 61792019
Deactivate
Refresh License
Activate Manually
Activate Online
Exit

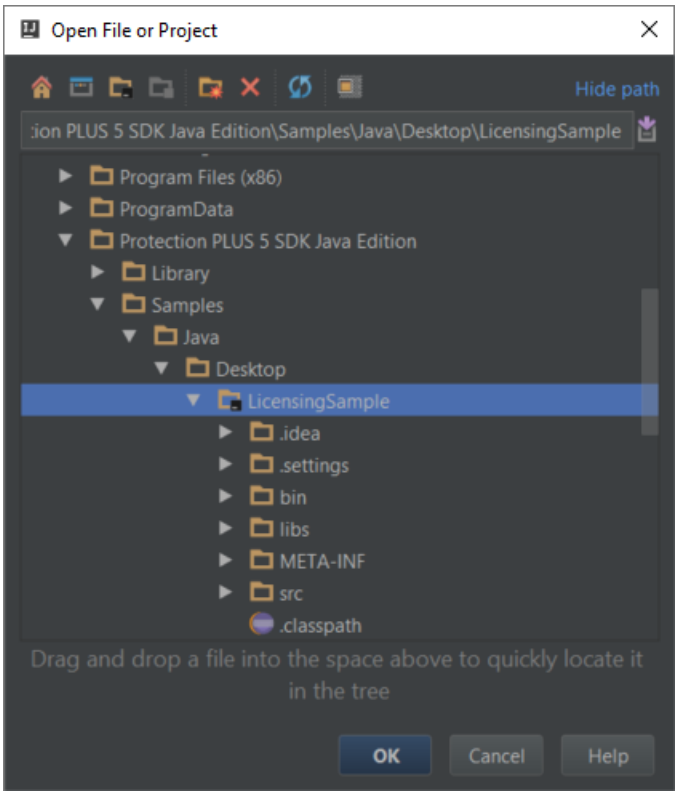
PLUSNative Java Desktop Licensing Sample

Step 1 – Opening the Licensing Sample Project

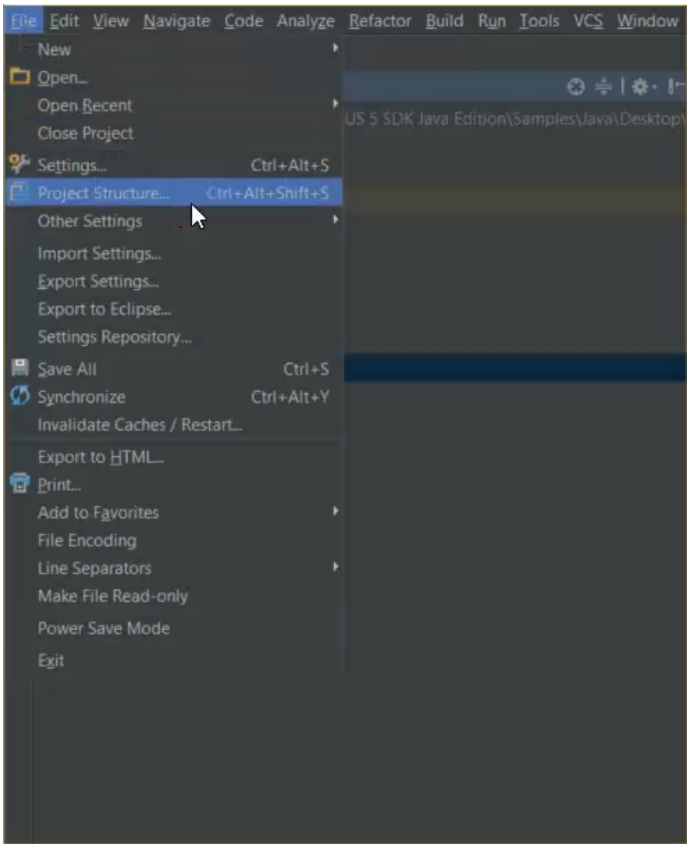
If you haven't already done so, download the Protection PLUS 5 SDK Java Edition zip file (ProtectionPLUS5Java.zip). You will typically acquire a link or button to download this zip file in an email if you are evaluating, or by logging-in as an existing customer at **www.softwarekey.com**. After downloading this zip file, extract its contents to a directory / folder of your choosing, and take note of the location you selected.

Next, open the LicensingSample project into your Java IDE of choice. As there are many Java IDE applications to choose from, this tutorial will only demonstrate using the IntelliJ IDEA Java IDE.

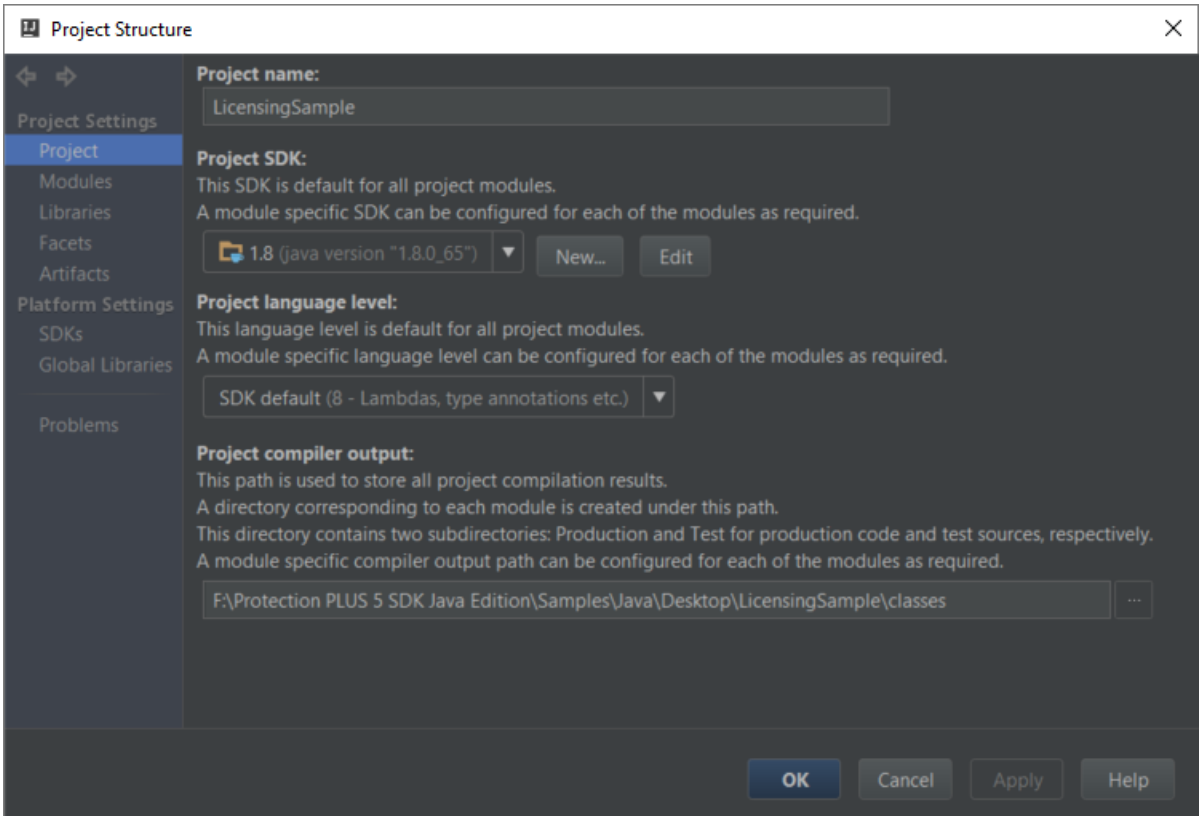
From the Menu choose File/Open. Browse to the *\Samples\Java\Desktop\LicensingSample* sub-directory located where you extracted the Protection PLUS 5 SDK Java Edition zip file.



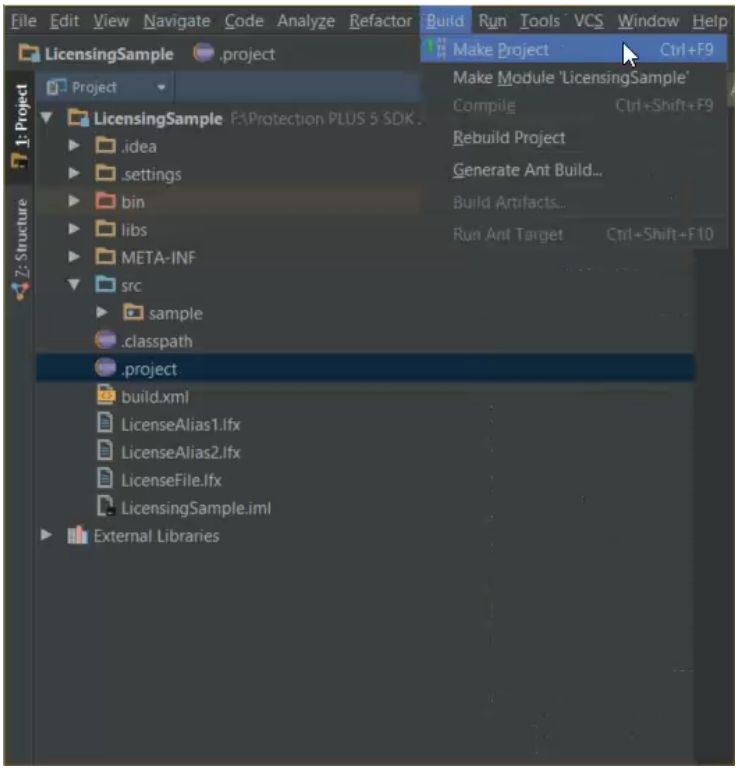
Next, we will setup the Java SDK Version. From the File menu select Project Structure to show the Project Structure settings.



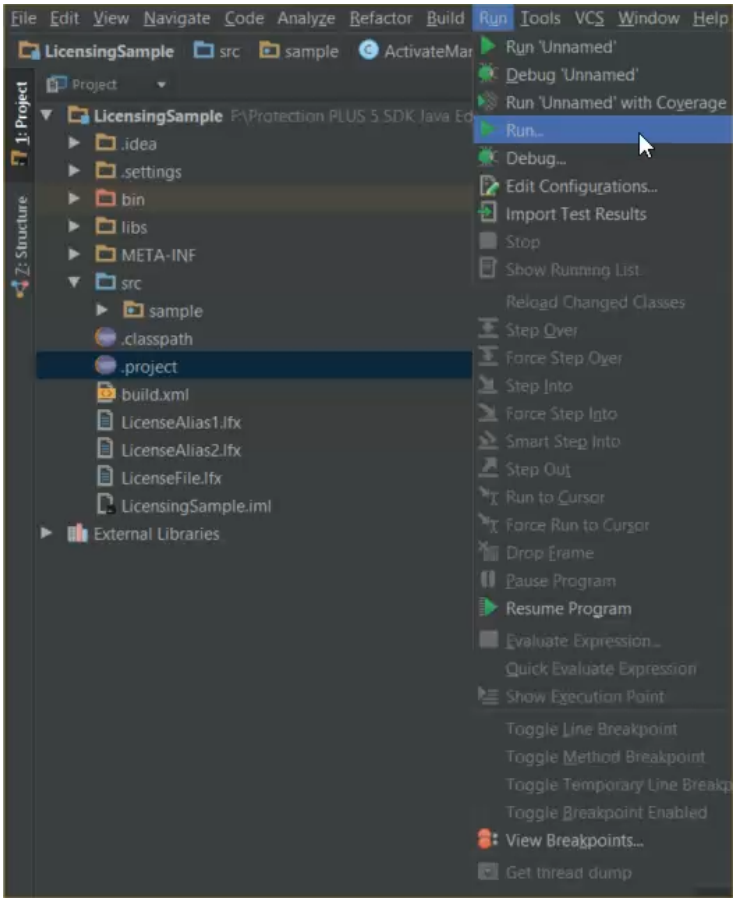
On the left click Project then on the right select your desired Java SDK version in the drop-down and click the OK button.



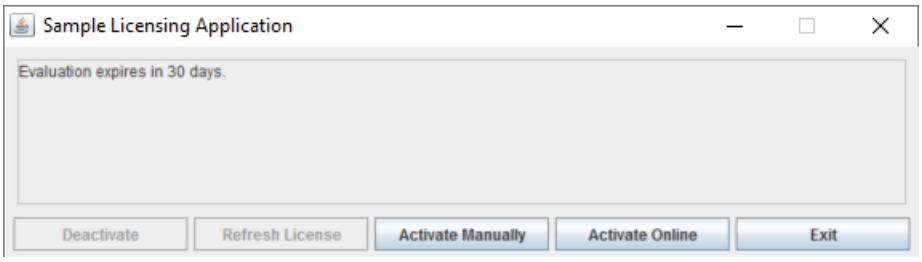
From the menu select Build/Make Project.



Once the project has successfully built, select Run/Run from the menu.

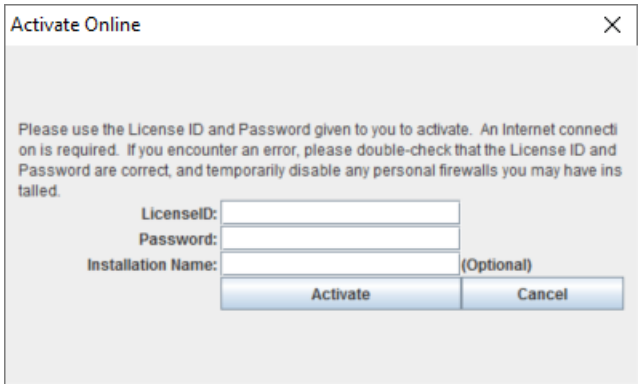


You will see the main LicensingSample dialog which will show the sample is in a 30-day evaluation mode.



Step 2 – Activating automatically with Instant SOLO Server

To activate the Sample Licensing Application automatically using SOLO Server, press the Activate Online button. A new dialog will prompt for a License ID and Password (both are required). Specifying an installation name is optional but helpful when performing multiple installations for different machines and/or users.

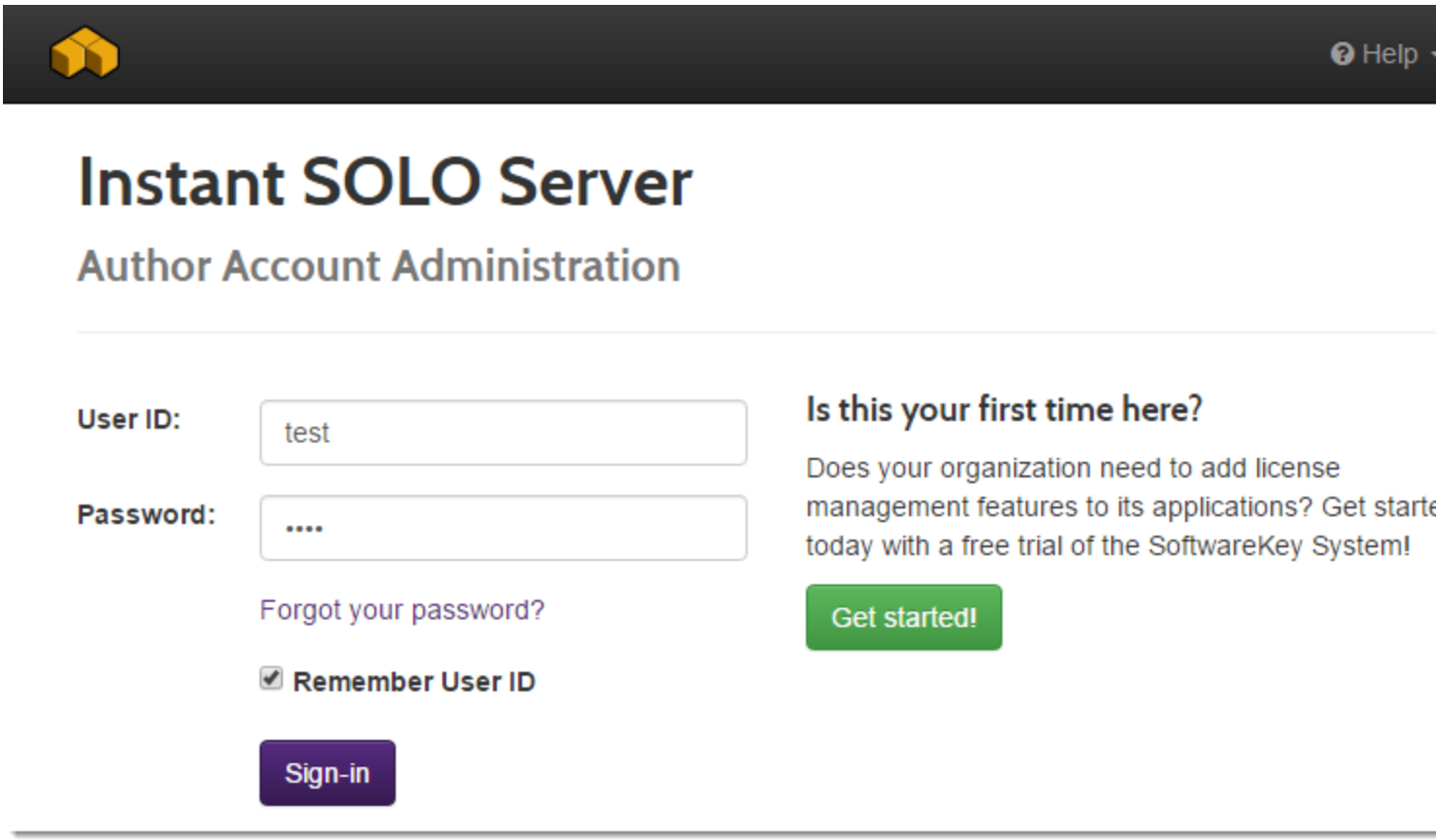


>

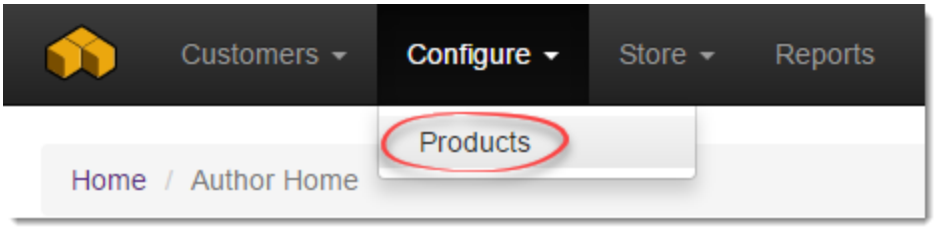
For ease of use, all of the Protection PLUS 5 SDK sample applications use a generic Test Author account with its own Encryption Key ID. We will use this Test Author account on Instant SOLO Server to generate the License ID and password necessary to activate the Licensing Sample

To **log into the Instant SOLO Server Test Account**, use the Test Author credentials given below:

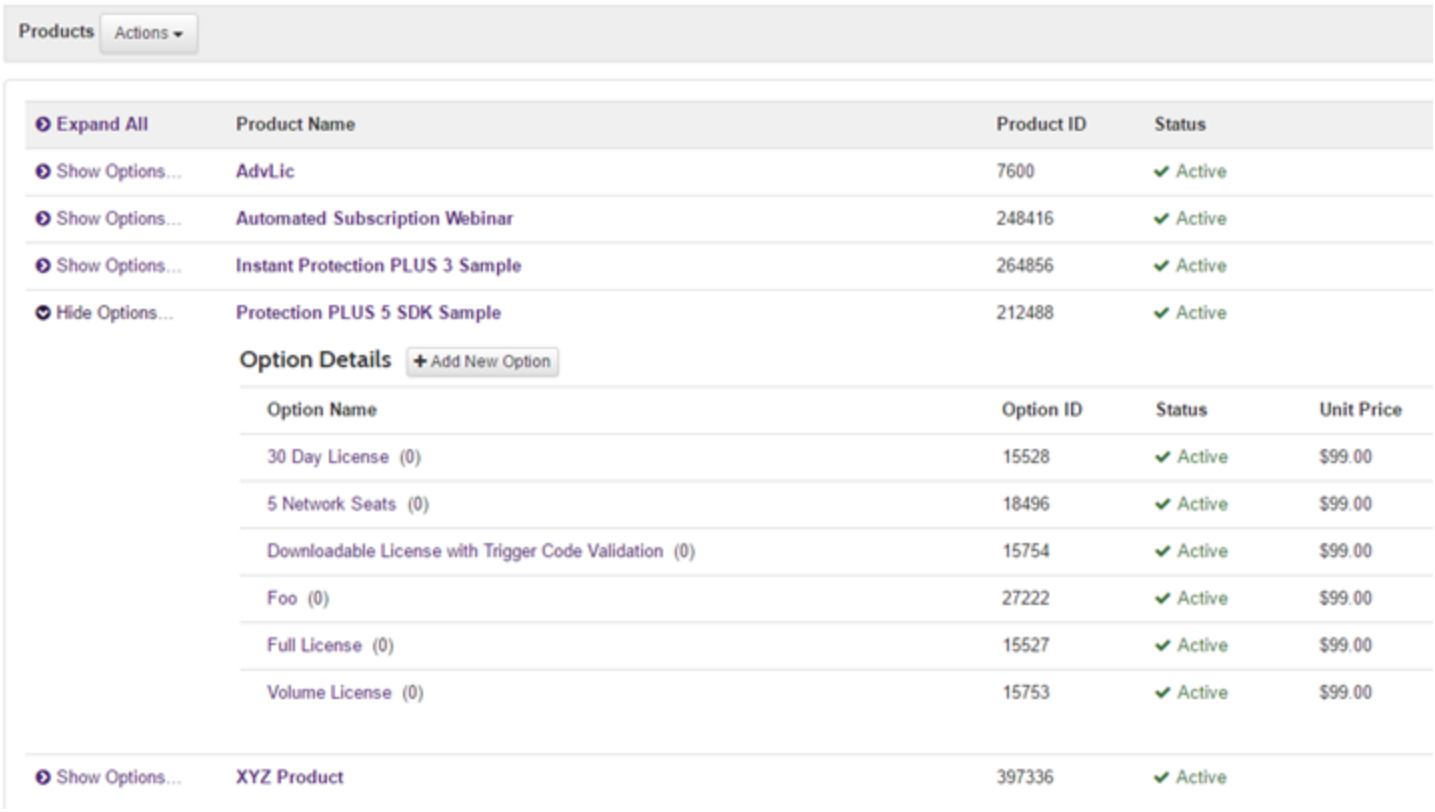
Login ID: test
Password: test



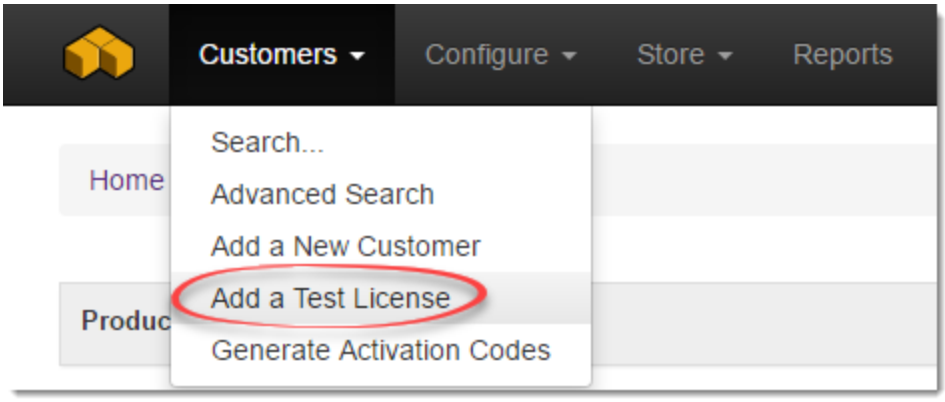
Once logged in, go to the menu *Configure / Products*.



The Product List page will show all of the products available to the Test Author. Expand 'Show Options' beside the Protection PLUS 5 SDK Sample product. The Option Details show different Product Options for activating the Licensing Sample application. For this tutorial, we will generate a license for the Full License.

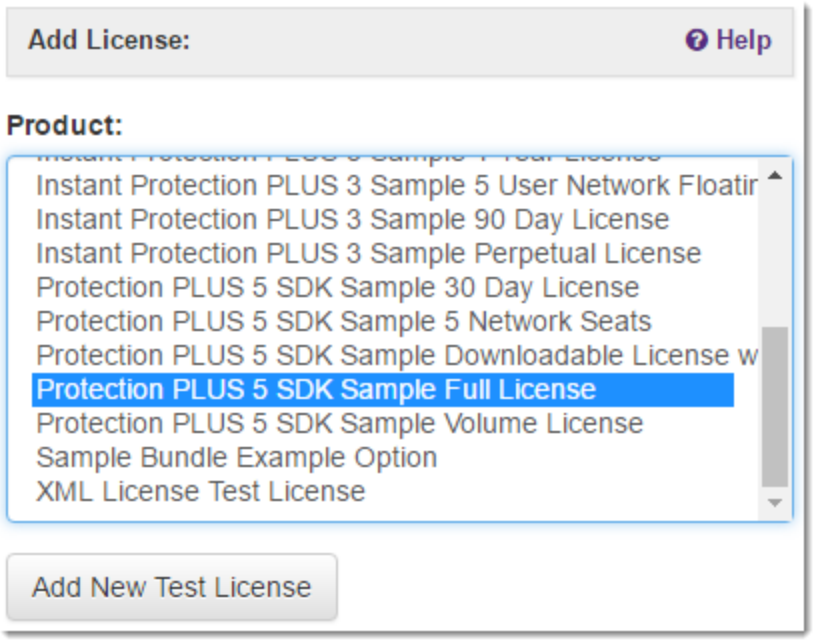


To add a test license for this product option, go to the menu *Customers / Add a Test License*.



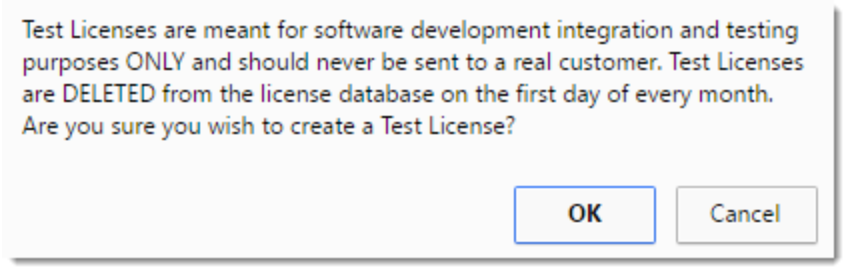
">

Select the Protection PLUS 5 SDK Sample Full License and click Add New Test License.

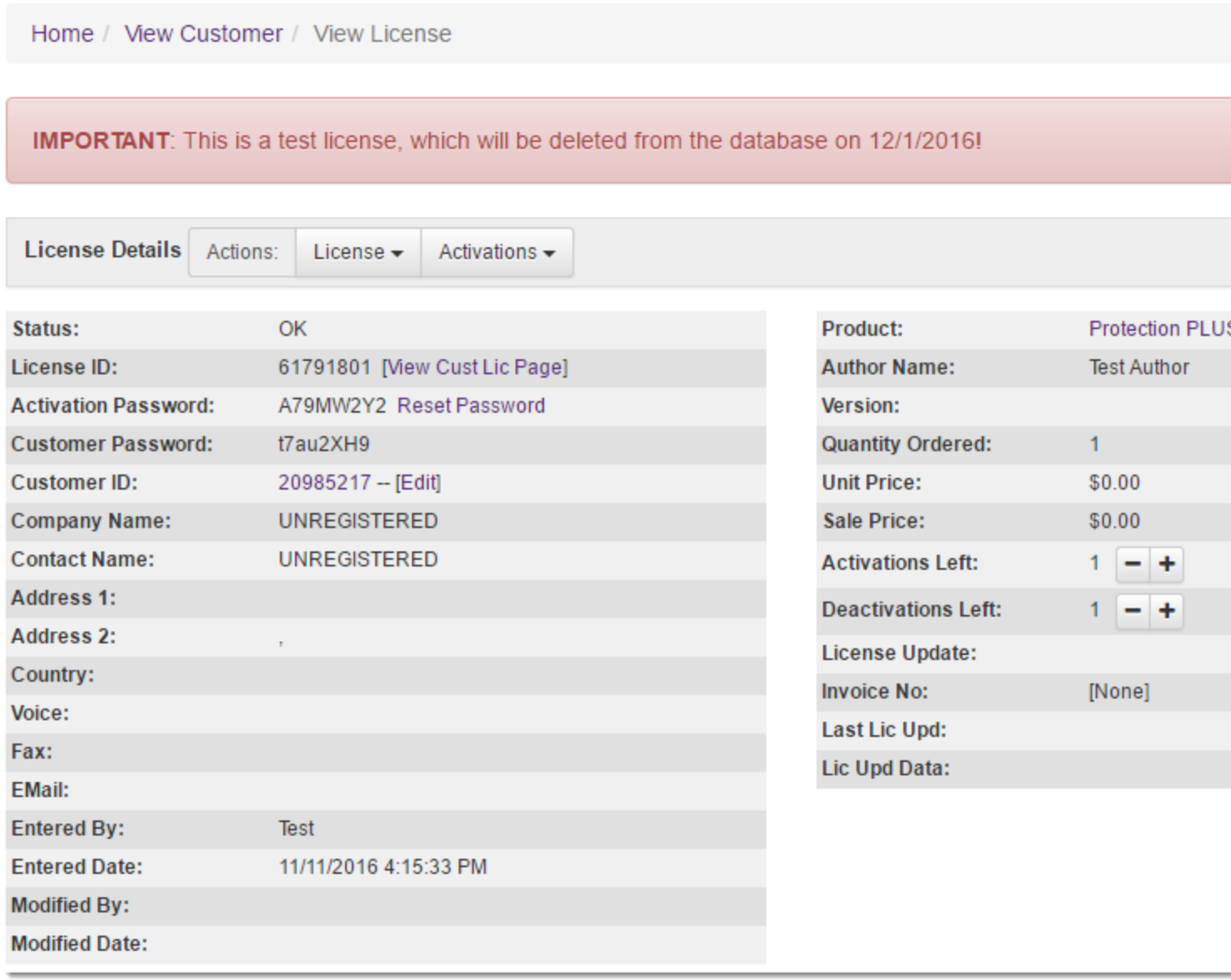


Important:

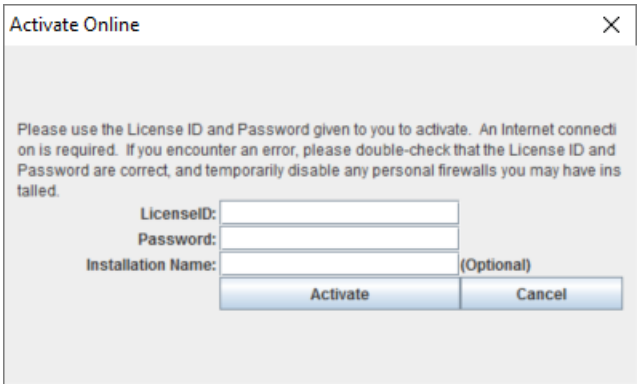
Test Licenses are meant for software development integration and testing purposes **ONLY** and should never be sent to a real customer. Test Licenses are **DELETED** from the license database on the first day of every month.



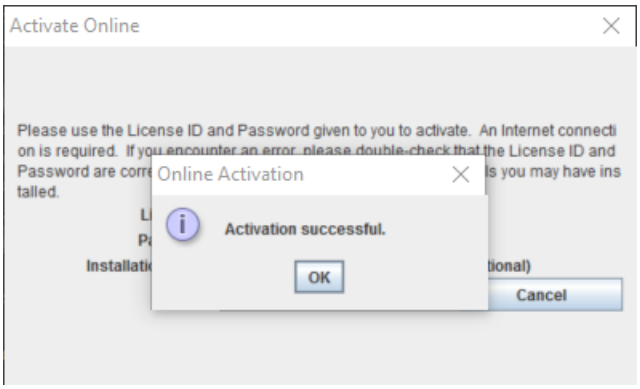
The Test License has now been created. We now have a License ID and Activation Password to continue testing the Licensing Sample.



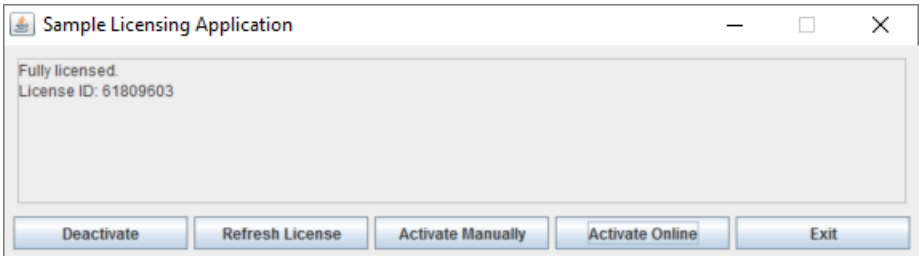
Return to the activation prompt for our Licensing Sample. Enter the License ID and Password in their respective fields and, optionally, an Installation Name. Press Activate to communicate with Instant SOLO Server for license validation.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.

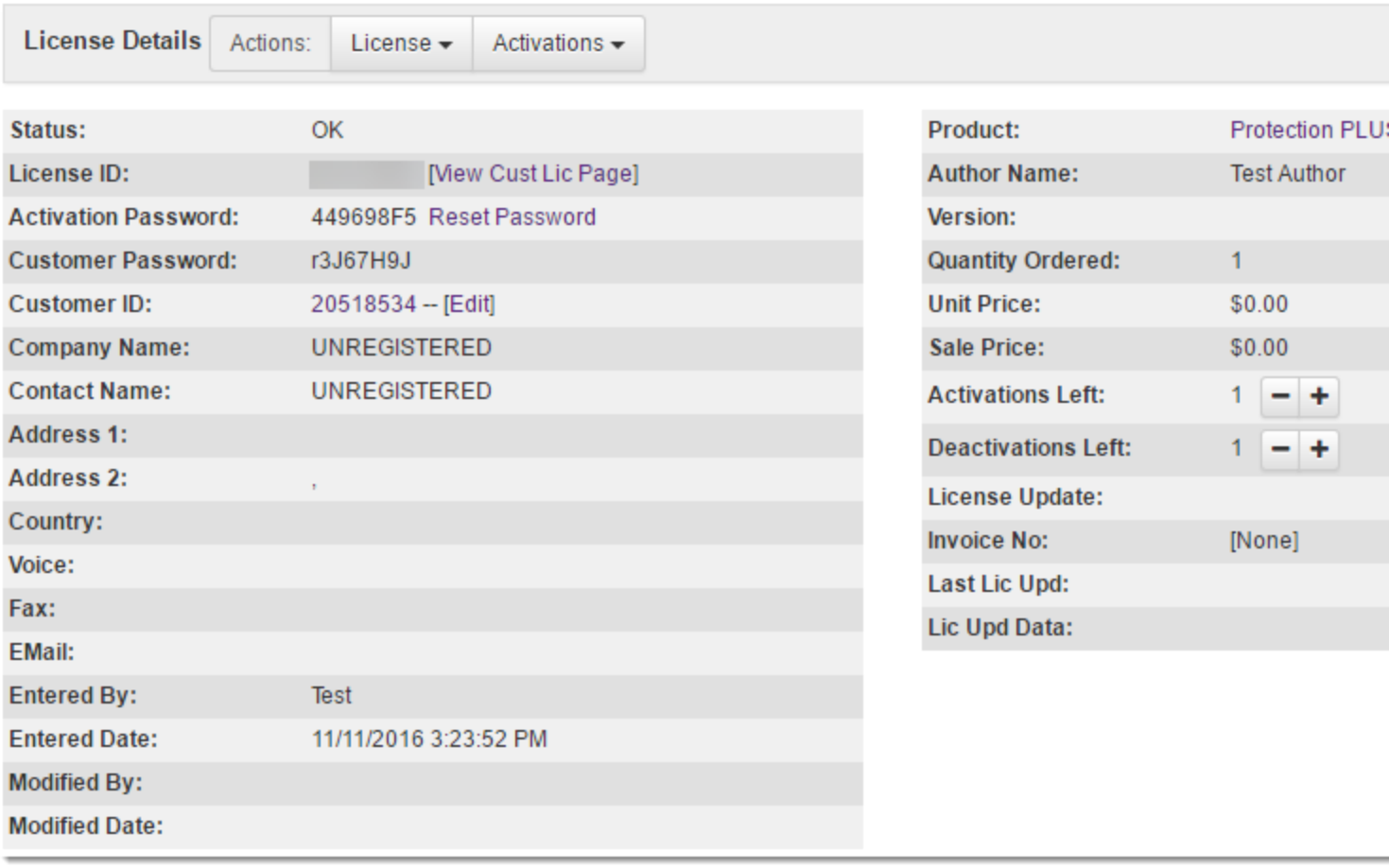


The License Status will now show as Fully Licensed.

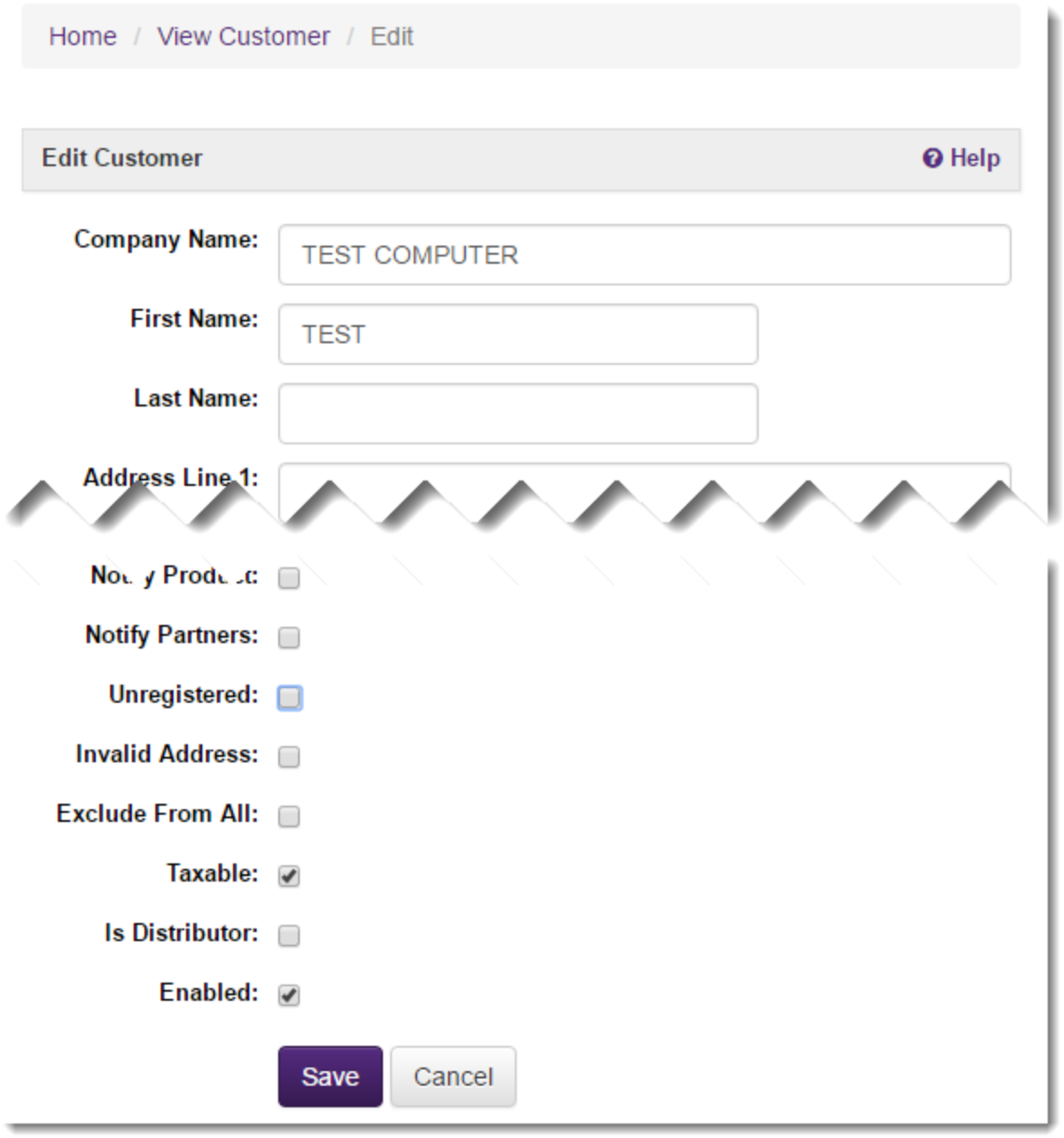


Step 3 - Refresh a License

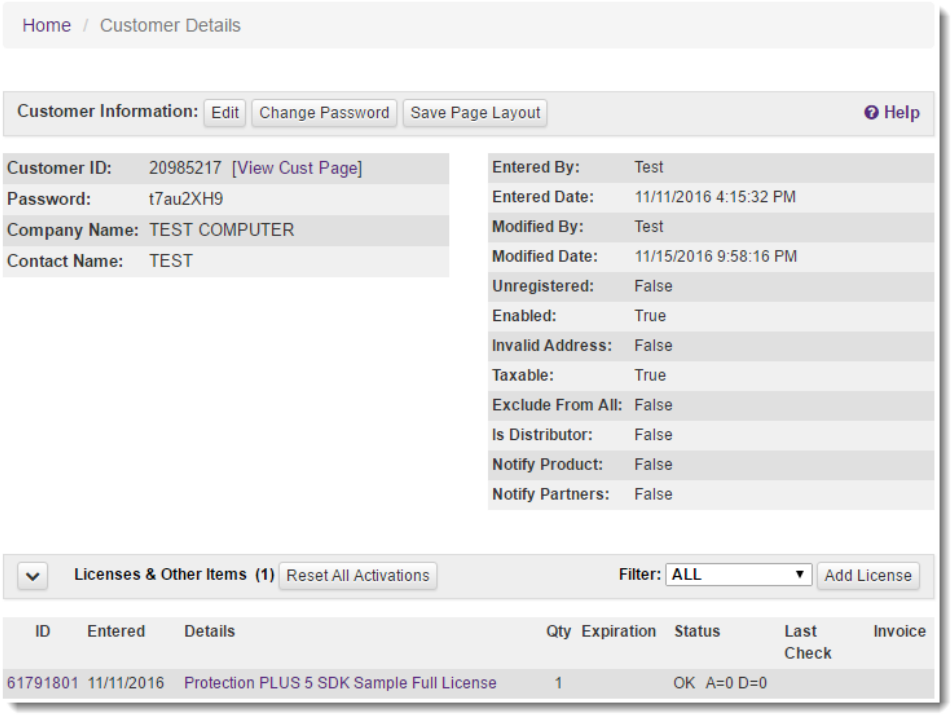
A license can be refreshed with updated information sent from Instant SOLO Server. We can modify customer details from the License Details page on Instant SOLO Server. On the License Details page, click Edit, located to the right of the Customer ID.



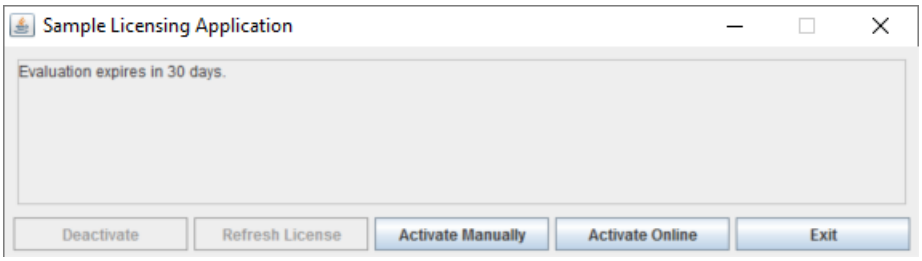
Modify the Company Name and additional fields as you please. Uncheck the check box setting to the right of Unregistered. Scroll to the bottom and click Save.



After the Customer Details have been saved, the page returns to the Customer Information page. This page includes customer details we have edited in addition to licenses and previous orders.

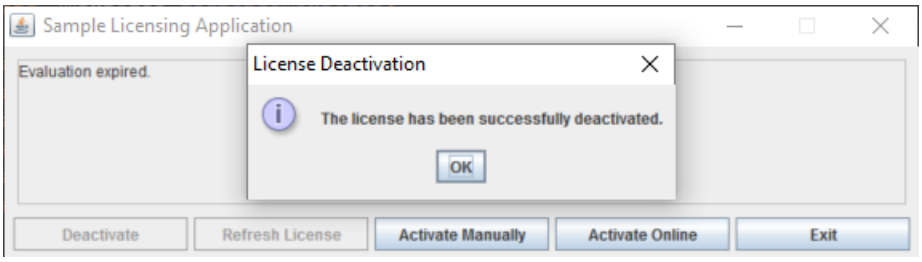


Return to the Licensing Sample Main dialog to proceed with a license refresh. Press Refresh License. If the license successfully refreshed, the License Status will now update with the modified customer details.

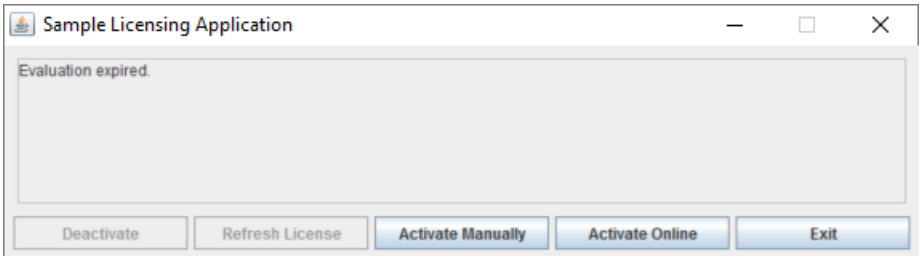


Step 4 - Process a Manual Activation

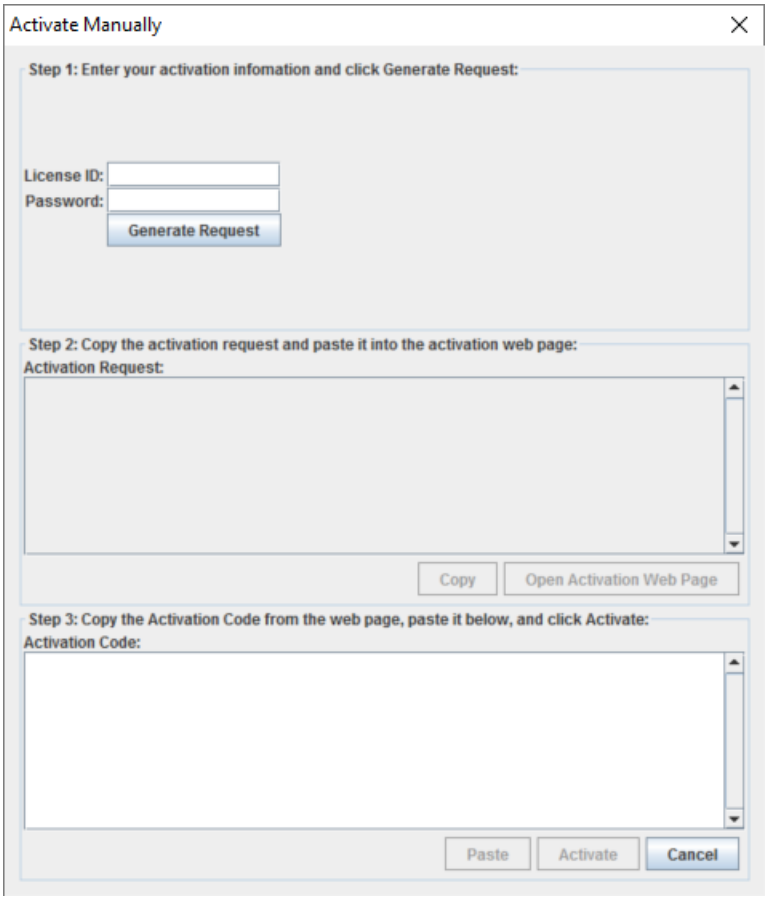
If you previously activated the Licensing Sample online using Instant SOLO Server, you will need to deactivate the license before testing manual activations. In the Main dialog of the Licensing Sample press Deactivate. A confirmation will appear if the license was successfully deactivated.



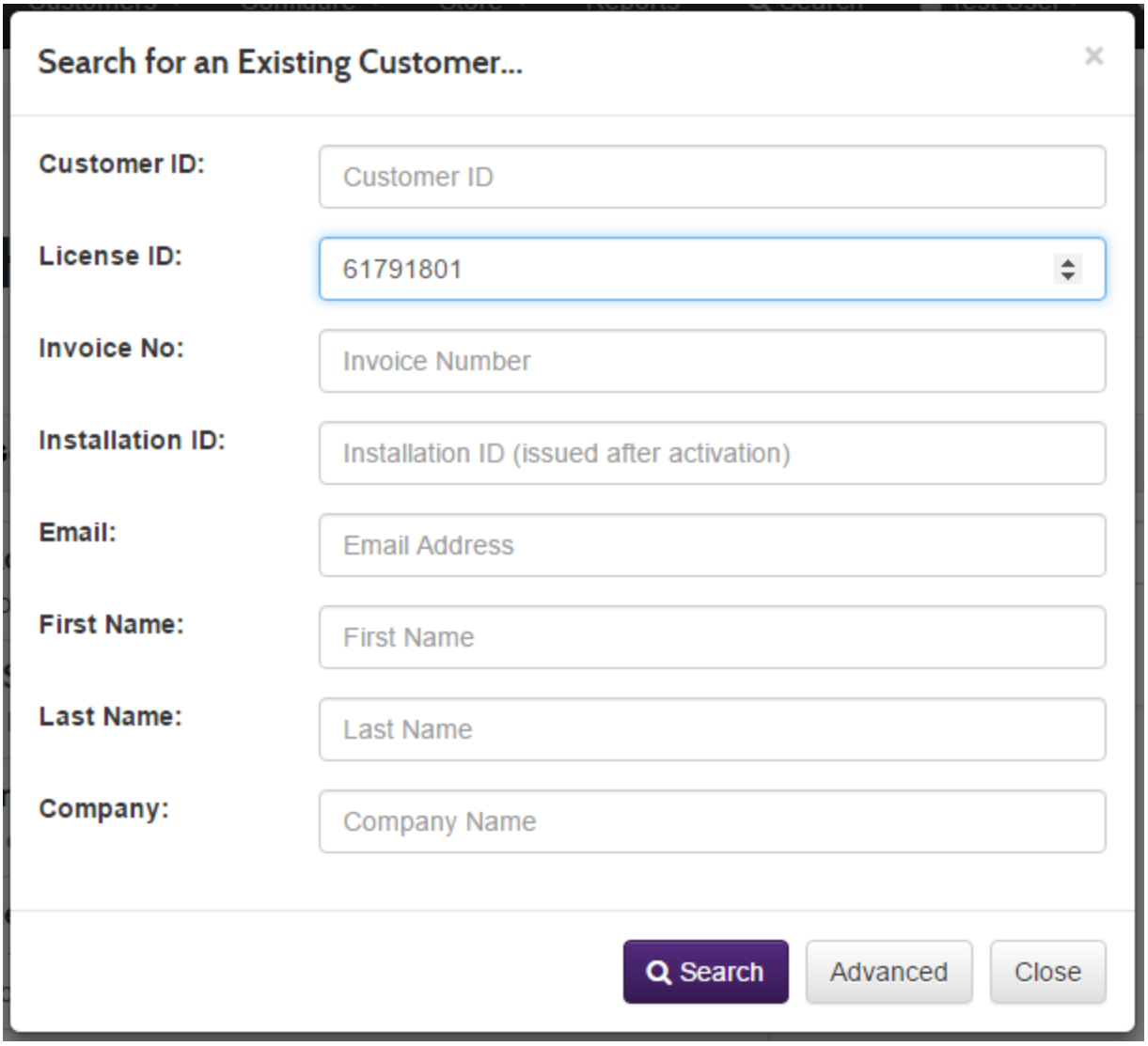
The License Status will now show "Evaluation expired."



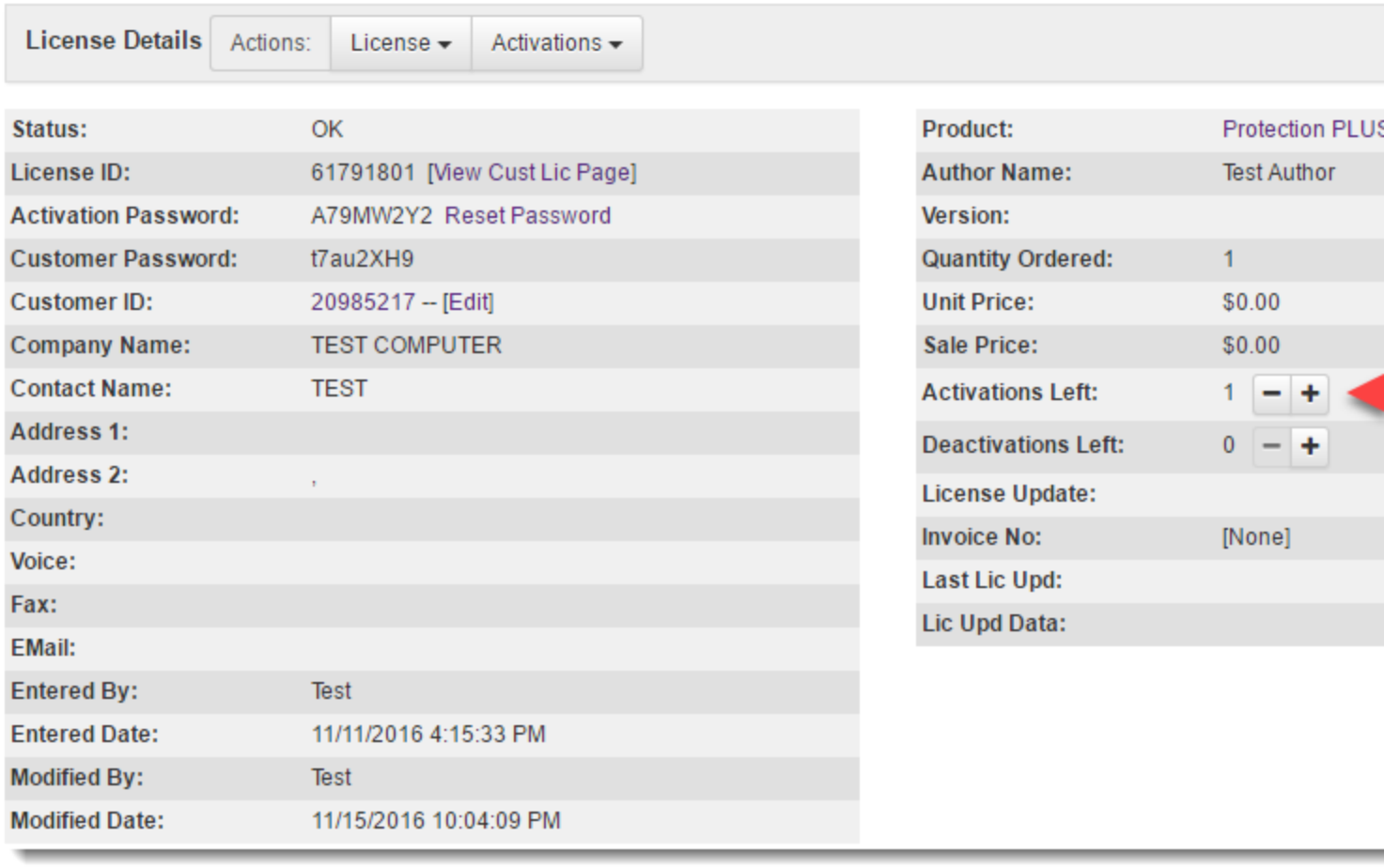
Press Activate Manually to proceed with testing a manual activation. A manual activation allows for activations from another computer or by e-mail.



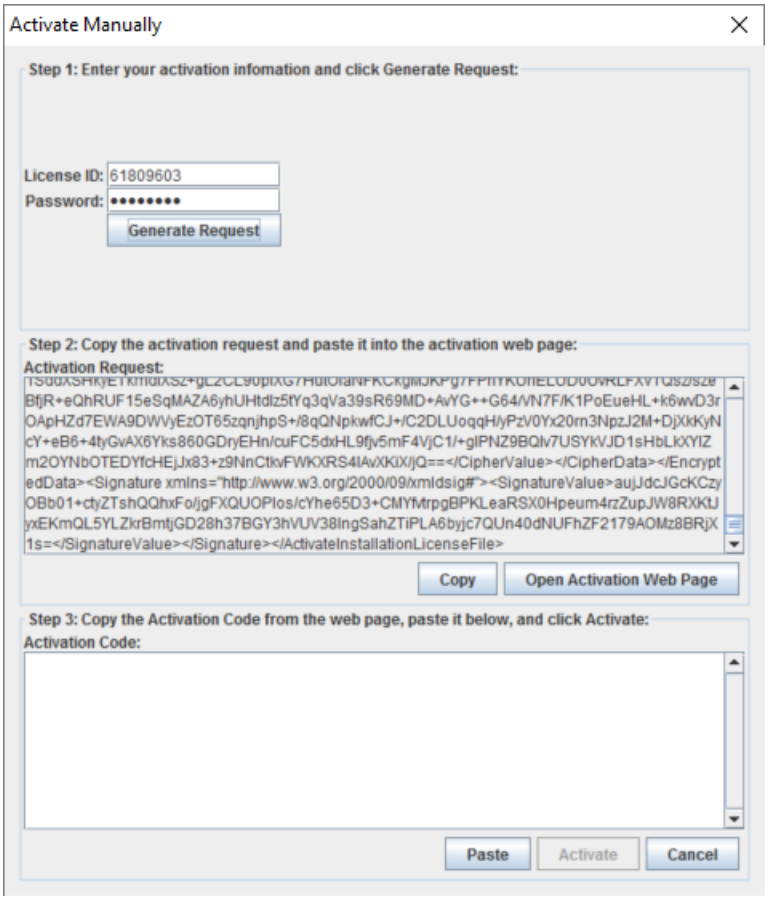
Return to Instant SOLO Server using the Test Author account. Perform a quick search for the License ID previously used to activate online (If you no longer have this number, proceed to create a new License ID as shown in Step # 2).



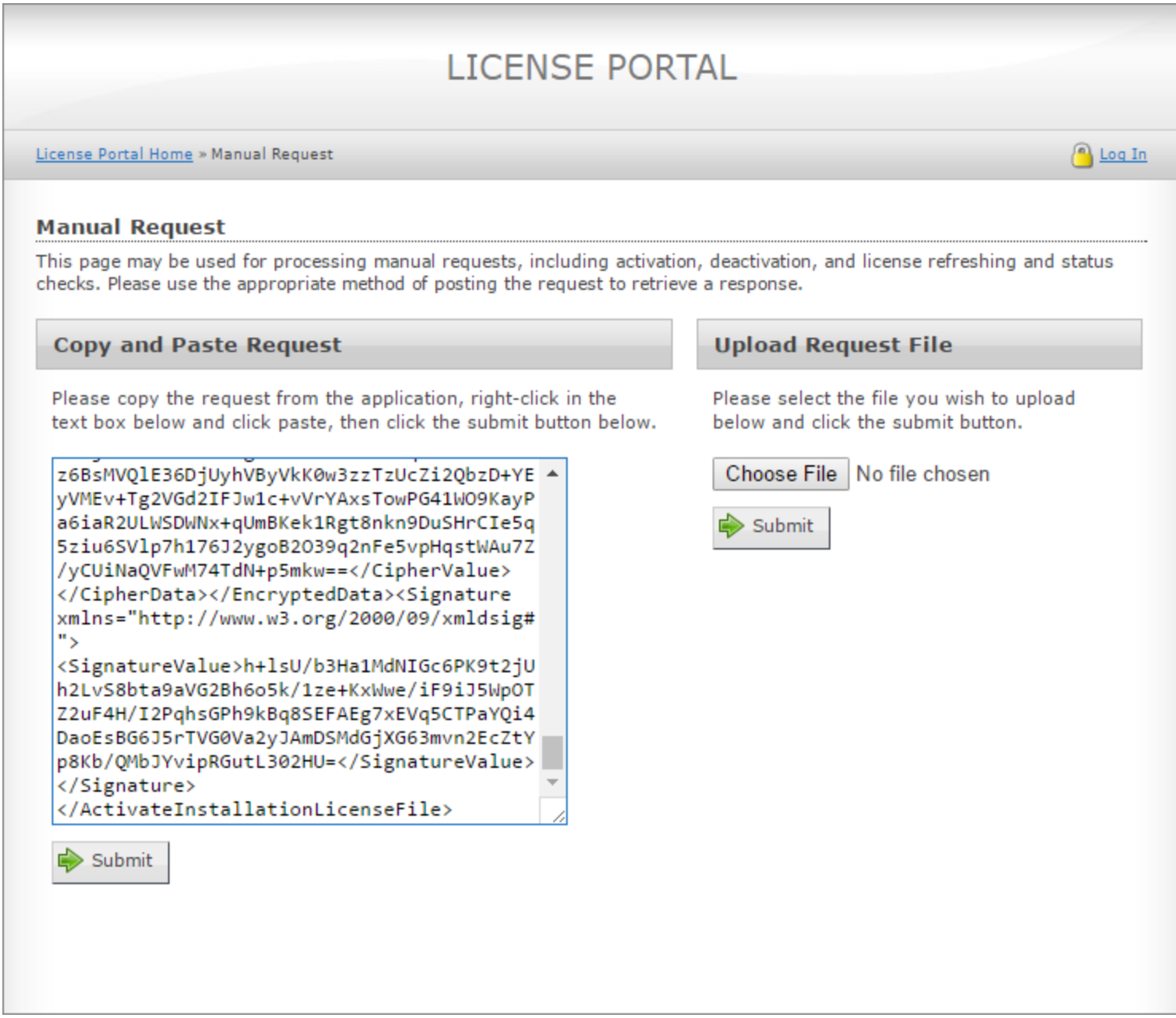
When we clicked on deactivate license, Instant SOLO Server automatically incremented the amount of activations left by 1. This setting was predefined in the Product Option ID we selected for testing. If you need to manually increase the number of activations left, you can increment this number by clicking the + button to the right of the Activations Left field.



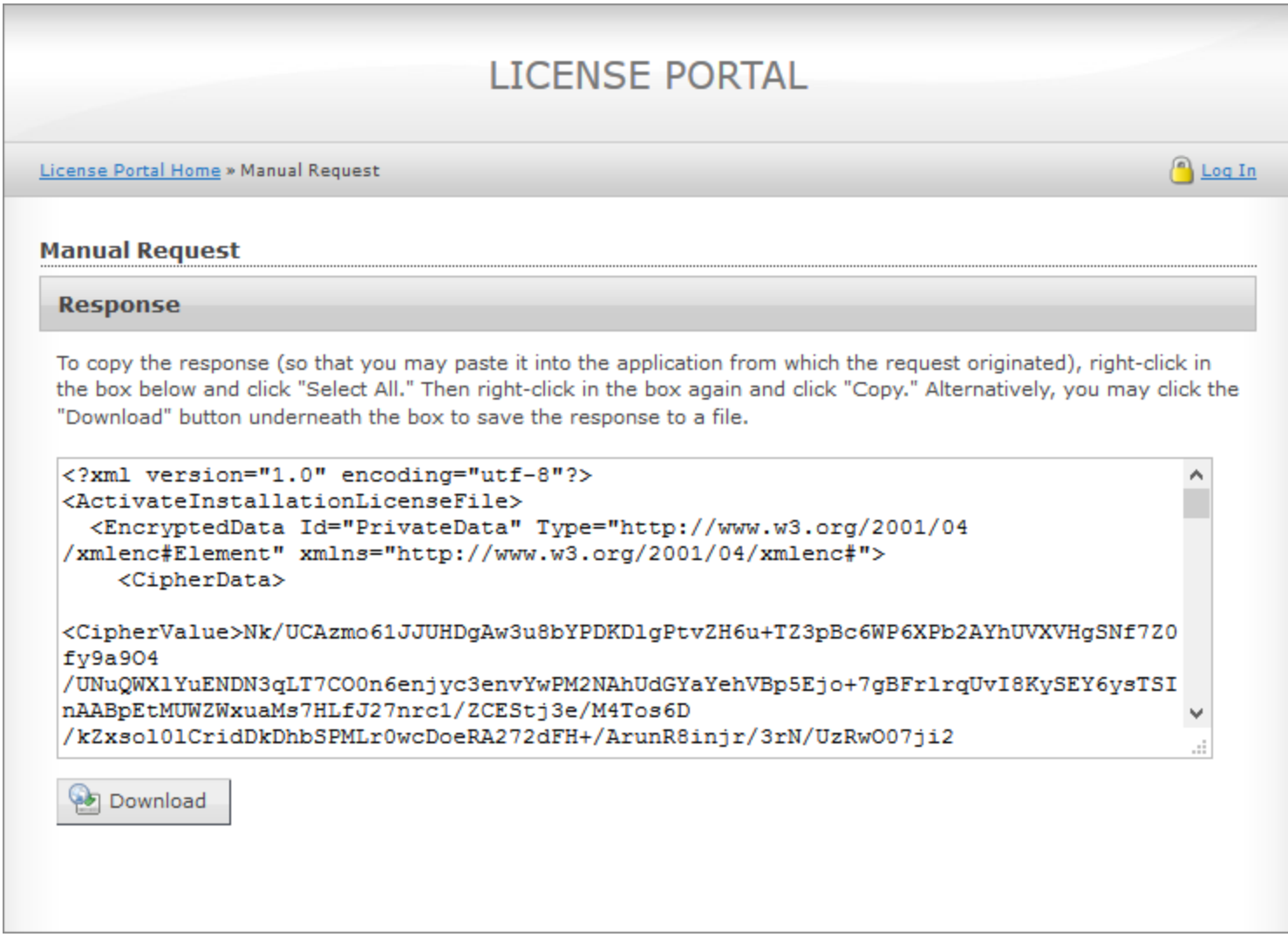
Return to the Activate Manually screen for our Licensing Sample. Enter the License ID and Password in their respective fields and press Generate Request as it will then be enabled.



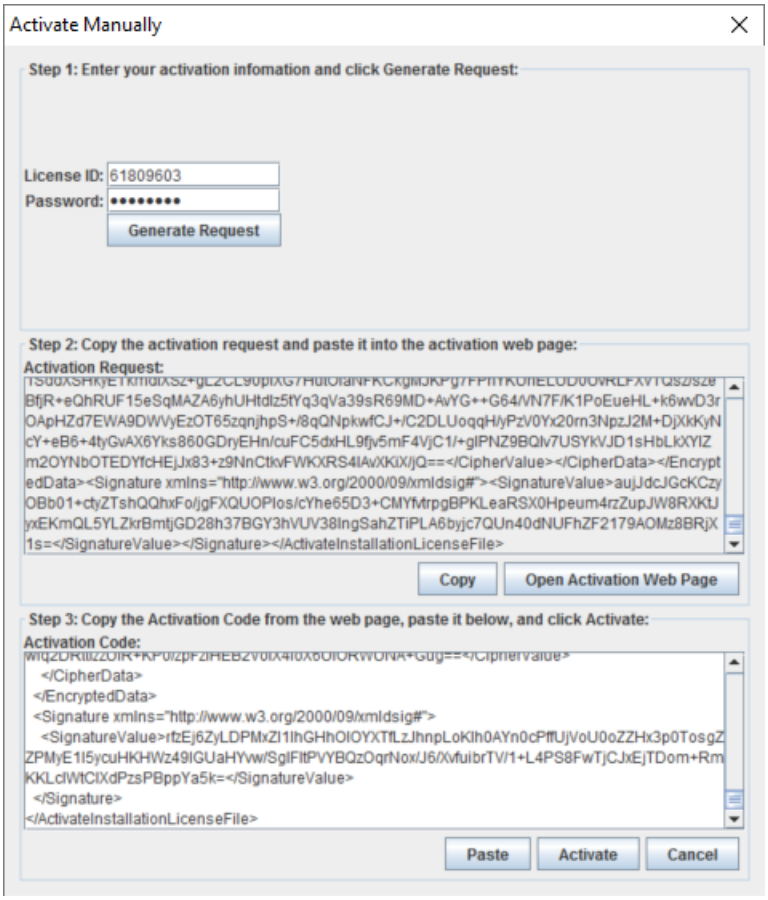
This dialog will generate an encrypted activation request to be processed by the activation server. Click Copy to copy this data to the clipboard. Next, click Open Activation Web Page. This web page is accessible from another computer with access to the Internet in the event that an offline workstation needed to be activated. Once you arrive to the License Portal page, paste the encrypted activation request data, and click Submit.



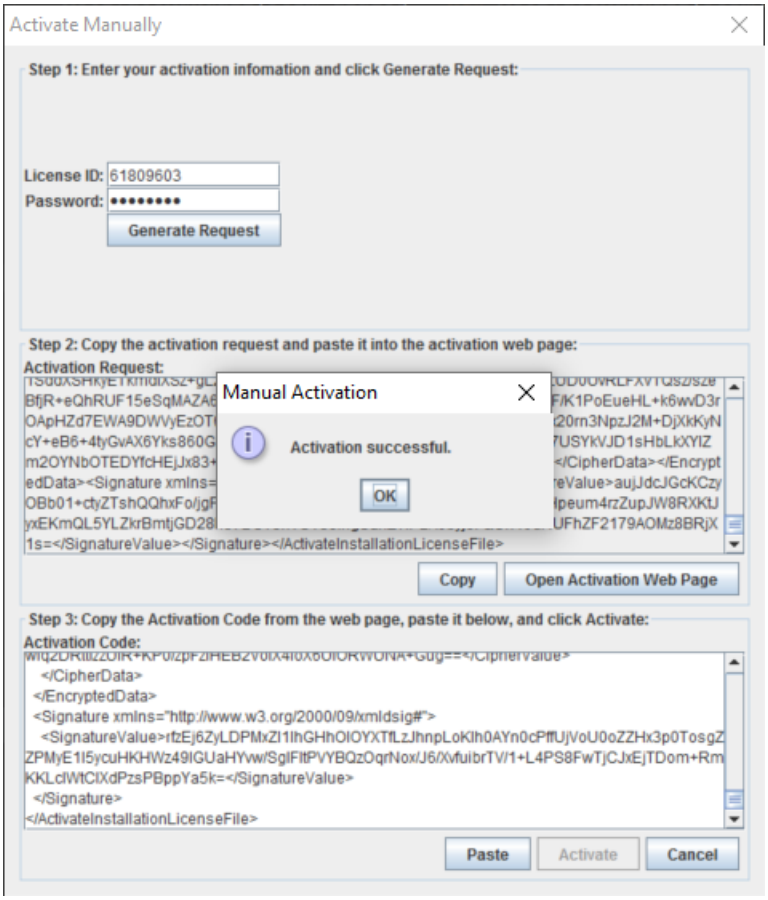
The License Portal will now generate an encrypted response. Copy this data to later paste in the Activation Code field of the activation dialog.



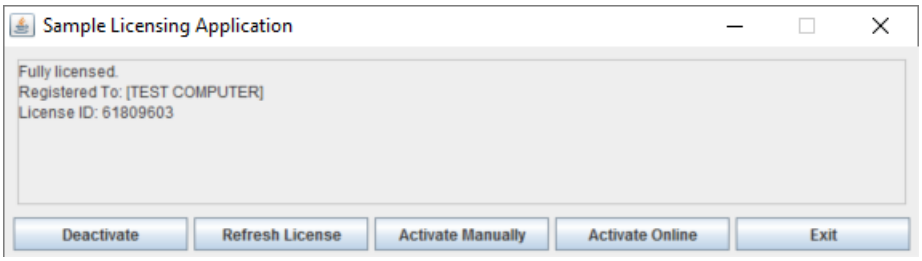
Return to the activation dialog and paste the encrypted response into the Activation Code field and click Activate.



Upon successful activation, a confirmation dialog is shown. If the activation fails, please verify the information entered with the license details on Instant SOLO Server.



The License Status will now show as Fully Licensed.



SOLO Server Overview

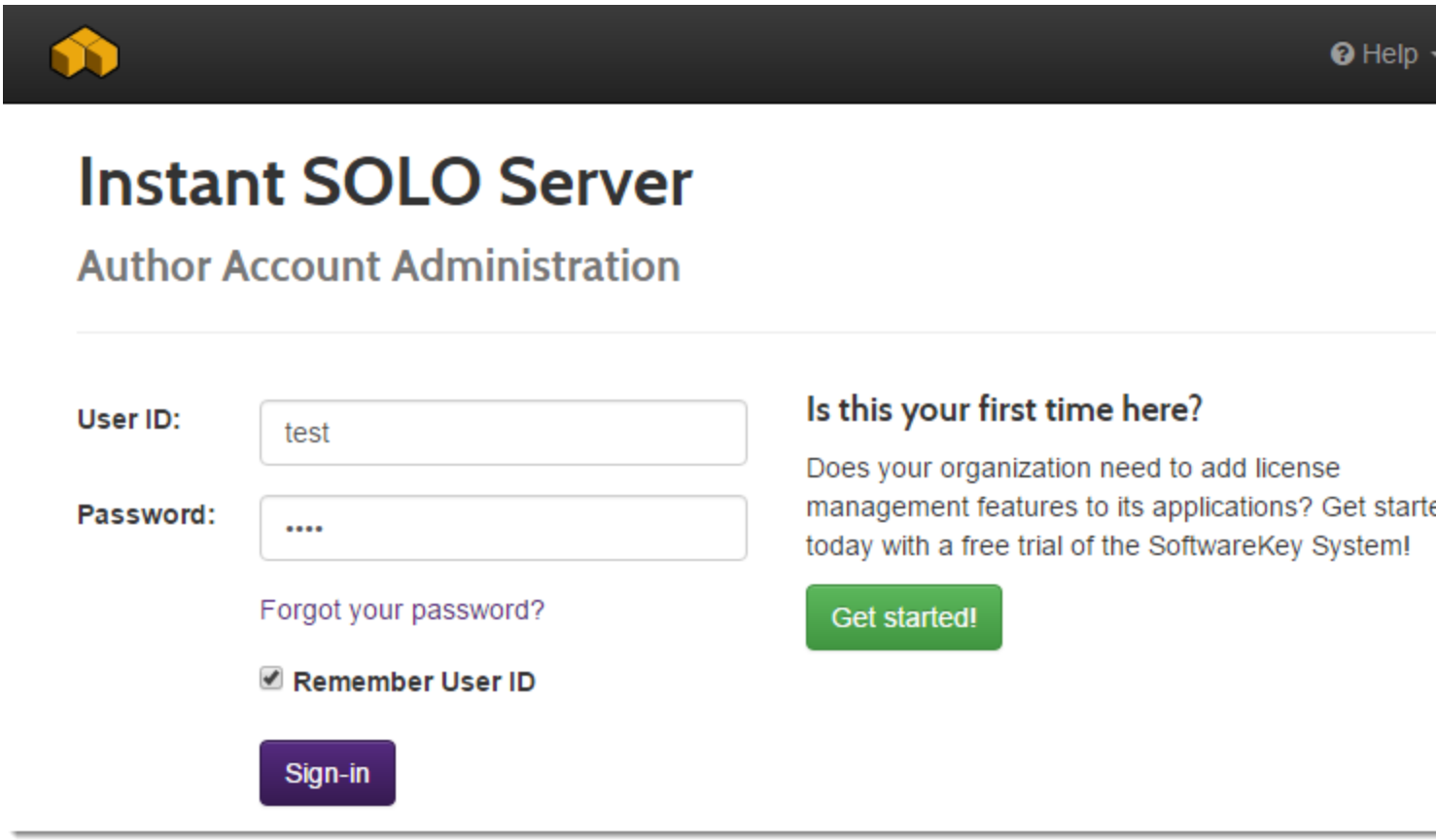
SOLO Server is power packed with features including **Electronic License Management™** (ELM) allowing you and your customers to activate and manage their software 24 hours a day while you maintain control. ELM is the must-have luxury feature for software companies who want to automate their online software business, both with or without integrated e-commerce. **Learn more** today, and check out the most popular **features**! We recommend using SOLO Server, but you can also use **License Manager** to manually activate customers offline although other ELM features are not available without SOLO Server.

Using SOLO Server

Evaluation

Testing integration with SOLO Server is a snap with Protection PLUS 5 SDK. Since the Protection PLUS 5 SDK samples are all pre-configured to work with a generic Instant SOLO Server test account, you can start by signing-in to its web interface at **https://secure.softwarekey.com/solo/authors/** using the following login credentials:

- User ID: test
- Password: test



Then, **add a test license** to get a License ID and Password.

Going Live

Once you have tested the sample applications, **sign up for your own SOLO Server trial account**. This is necessary to allow your software to be activated online and to use the optional e-commerce features.

Important

Before you distribute production copies of your application, it is imperative that you **update your application for your SOLO Server account**.

Once your account is configured, your application is **configured for your account**, you are ready to sign-in to SOLO Server, **configure your products**, and request account activation to start licensing your application!

Using your own SOLO Server account

If you have purchased Protection PLUS 5 SDK and signed up for an Instant SOLO Server account or purchased SOLO Server, you will need to update your application so it no longer uses our generic Instant SOLO Server test account data and instead uses your own Product ID, Encryption Key Data, and potentially your own web service endpoints.

Product Configuration

The first step is to log into your account and **create a product and a product option**.

Important

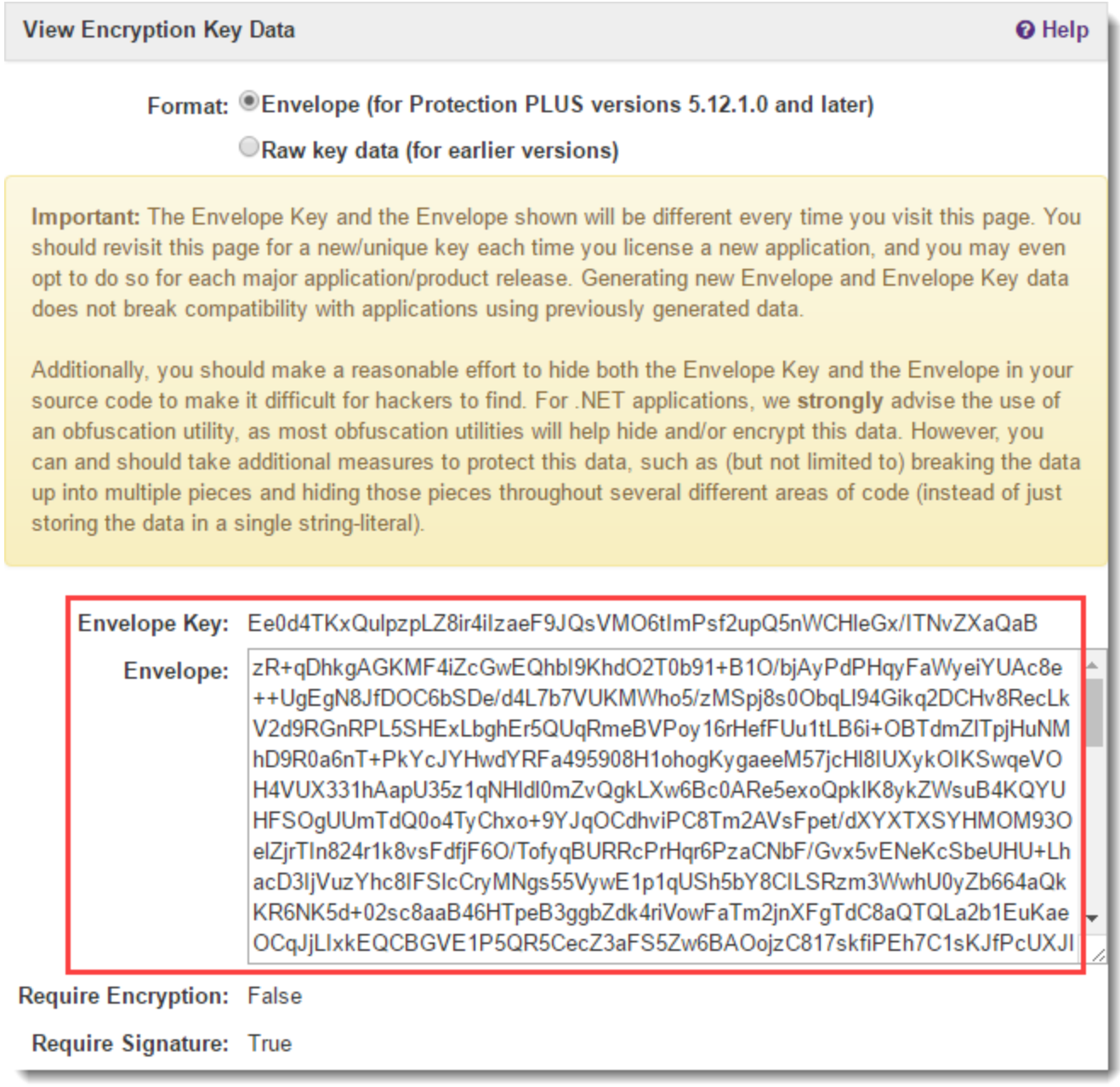
When configuring your product options, Protection PLUS 5 SDK requires the "Issue Installation ID" option to be checked!

Take note of your Product ID, which can be viewed by using the menu *Configure / Products* . You will add this Product ID to your source code so it can be validated during activation.

Encryption Key Data

Next, you will need to update your application to use your account's encryption key data. This ensures that only licenses created by your SOLO Server account will be able to activate your software. See our **Cryptography and Security** topic for more information on how this works.

To get this data, go to the menu *Configure / Products*then use the Actions dropdown to choose *View Encryption Key Data*. The **Format** needs to be set to Envelope, and the data in the **Envelope Key** and **Envelope** fields needs to be set in your source code.



Important

For your convenience, the Protection PLUS 5 SDK sample applications are configured to use a generic test SOLO Server account encryption key data for your evaluation purposes. This data is **not intended for product distribution**, and any license created with this test account is automatically deleted. Furthermore, using the gen-

eric test account's encryption data in your application would mean anyone could use that same data to manage/bypass your application's license.

In the PLUSManaged sample applications, you will find the encryption key data in the LicenseConfiguration class under the Encryption Settings region.

For information on how to set the encryption key data (and the Product ID), view the topic on **Creating the License Implementation** for PLUSManaged, or the topic for **Configuring the API Context** for PLUSNative.

Web Service Endpoints

If you are using Instant SOLO Server Shared URL, you do not need to make any changes here since it is already using `https://secure.softwarekey.com/solo/` in the samples. In all other cases, following the instructions below depending on which SOLO Server package you are using. In the PLUSManaged sample projects, you would make these changes in the WebServiceHelper class. For additional information on calling the web services, view these topics:

- **PLUSManaged: Calling SOLO Server XML Web Services**
- **PLUSNative: Calling SOLO Server XML Web Services**

Instant SOLO Server Custom Shared URL

Update `https://secure.softwarekey.com` to `https://[your brand].softwarekey.com` where `[your brand]` is your sub-domain.

Instant SOLO Server Dedicated URL

Update `https://secure.softwarekey.com/solo/` with `https://[domain]/[SOLO root]/`, where `[domain]` is your fully-qualified domain name and `[SOLO root]` is the root directory for Instant SOLO Server.

SOLO Server (self-hosted)

replace `https://secure.softwarekey.com/solo/` with `https://[domain]/[SOLO root]/`, where `[domain]` is your fully-qualified domain name and `[SOLO root]` is the root directory for SOLO Server.

Important

Test all of the web service endpoints in your application thoroughly to ensure everything is functioning properly with the correct endpoint URLs, the correct Product ID, and correct encryption key data.

SOLO Server Product Configuration

Before you can begin issuing and activating licenses, you must define and configure the products and product options in SOLO Server which will be used to issue your application's licenses. This topic is designed to guide you through adding your first product option, and assumes you have already **defined your licensing requirements**. Additionally, the SOLO Server manual can be viewed by clicking the menu *Help / Manual* or *Help / Help for this Page* in the very top right of the administrative web interface.

Important

Before you distribute production copies of your application, it is imperative that you **update your application for your SOLO Server account**.

In SOLO Server, a *product* is defined to describe your application as a whole, and may contain one or many *product options*. For each type of license your application will support and require activation to use, at least one product option needs to be defined. In this guide, we will use an example product named "XYZ Product" with the following example licensing requirements:

- Self-issue 30 day evaluations.
- Require activation for a non-expiring license after the evaluation has expired.
- Support two different product editions for activated, non-expiring licenses, including an "Express Edition" (which includes 90 days of support and updates), or a "Standard Edition" (which contains more features, and 1 year of support and updates).

Creating a Product

1. Sign-in to SOLO Server. This can be done by clicking the "Log-in to SOLO Server" button or *Tools / Log-in to SOLO Server* menu option in [[[Undefined variable CSI.ProductName]]], or by clicking on the **Instant SOLO Server Login** link at the SoftwareKey.com website (for Shared URL authors).
2. Click the *Configure / Products* menu item then use the *Actions* dropdown and choose *Add Product Wizard*.

Create or select a Product...

This screen in the "Add Product Wizard" allows you to create a new Product, or add an Option to an existing Product.

When adding a new Product, you also have the option to copy all settings from an existing Product; however this does not copy the existing Product's Options, Upsells, or Updates.

Note that all steps after this one in the the "Add Product Wizard", a new panel will show at the top titled "Summary of current selections..." until you reach the end of the wizard (at which point this is all that is displayed). This panel provides a helpful summary of the selections you made as you progressed through the wizard, and is also very helpful in summarizing combinations of certain settings.

Configure your new Option...

This panel allows you to create a new Option from a blank template, or create a new Option that has all settings copied from an Existing option. When you choose to copy all settings from an existing Option, this will not copy Extended Pricing, Priority Codes, Shipping, Upsells, and Rules. If you need to make a copy of an existing Option that also copies these additional settings, use the **Duplicate Product Option page** instead.

The selection you make in "Type of Product" field should describe the nature of the item that is being sold. Always select "Licensed Software/Application" if you are selling a license that may be activated with SOLO Server via a supported licensing client (configured on the next screen). You may still configure download and shipping settings after selecting "Licensed Software/Application."

Configure your licensing settings...

Selecting your licensing client

You may be asked to select the licensing client you use.

Choose Protection PLUS 5 SDK as your licensing client.

- Protection PLUS 5 SDK licensing is based entirely on the needs of your software and how you **define your licensing requirements**. Consequently, you must select the Option Type and Trigger Code Number appropriate for the behavior defined in your application's source code.

Configure download settings...

This screen allows you to configure a file that may be downloaded via SOLO Server's **ESD features**, or a download page hosted on your company/product web site, which can be provided to the user from SOLO Server's **Customer License Portal** and **invoices**.

Configure e-commerce settings...

In this screen, you may configure basic settings for SOLO Server's e-commerce, such as the price and unit of measure.

Important

Setting the price to zero allows anyone to use the shopping cart to process an order without entering a credit card, even if you are not using the e-commerce engine of SOLO Server. It is strongly recommended that you set the price to a value greater than zero unless you specifically want to allow this or have configured a shopping cart rule to verify eligibility for a free upgrade.

Configure miscellaneous settings...

This screen allows you to configure miscellaneous settings, such as whether the new Product and Option are hidden and enabled.

Review and save

This screen provides a final summary of your settings and selections, and allows you to save your new Product and Option.

Important

When configuring your product options, Protection PLUS 5 SDK requires the *Issue Installation ID* option to be checked!

The product option configuration offers a plethora of configuration options not covered in this guide. Each of these options are documented in the SOLO Server manual, which you can see by clicking the Help link at the top of the page.

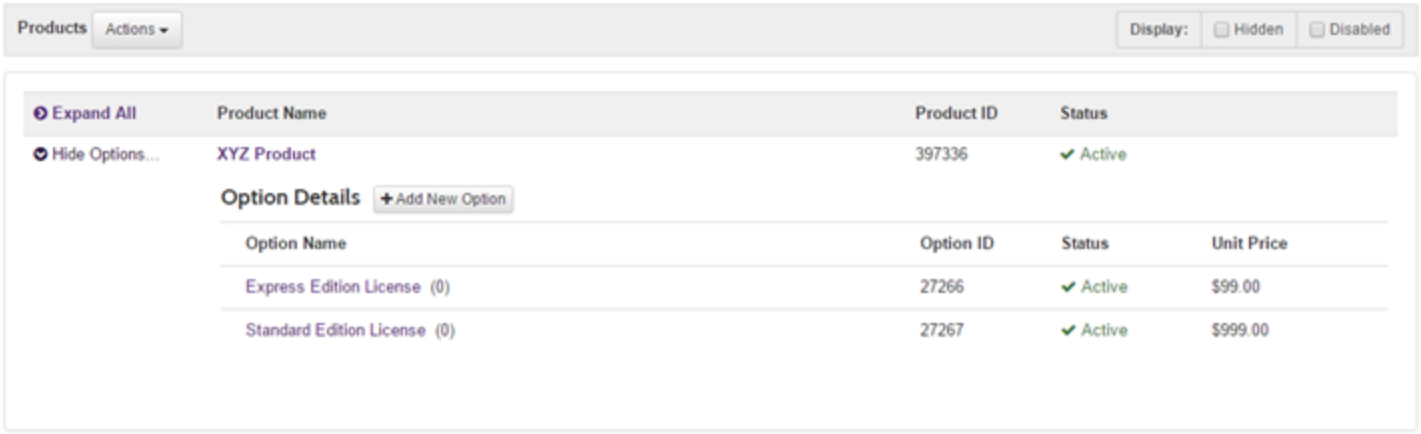
Creating Additional Options For Your Existing Product

It is possible to simply create additional product options the same way the first was created. However, it is generally a good practice to copy existing product options to create your additional product options. This is especially true when

using Protection PLUS 4 compatible activation codes, and when using the SOLO Server shopping cart. Consequently, it is strongly recommend to follow the steps below to create your additional product options:

1. Return to the product listing page if you are not already there. You can do this by clicking the *Configure / Products* menu item.
2. Click the *Actions* dropdown menu and select the *Add Product Wizard*.
3. Select *Add new Option to an existing Product*, select the Product, and then click *Load and Continue* at the bottom.
4. Select *Define a new Option starting from a copy of all settings from an existing Option*.
5. Select the Product that contains the Option you wish to copy, select the *Option to Copy*, and then click the *Copy* button.
6. Choose the *Type of Product*, enter a name for it, and then click the *Continue* button.
7. Make any appropriate changes to the licensing settings and click the *Continue* button.
8. Make any changes to the additional settings as necessary.
9. Once you get to the *Review and save* page, go over your chosen settings, and then click the *Save* button.

With everything configured as described in our example and steps above, the new Option will be added in your Product list/catalog as shown below.



Preventing Cross-Product Activation

When your application submits activation requests to SOLO Server, it can include the Product ID (232912in the example screen shot above) in the request. By doing so, SOLO Server will ensure that the License ID the user entered for activation is for the appropriate Product ID specified in the request. Since each Product ID is generally unique for each of your applications, this validation prevents licenses issued for one product or application (i.e. "XYZ Product") from being used to activate an entirely different product or application. If no Product ID (or a zero) is sent to SOLO Server, then SOLO Server will not perform this validation for you, and you will need to make sure you evaluate the Product ID and Product Option ID in the activated license to ensure the license is for the appropriate application.

Furthermore, the Product Option ID may also be included in an activation request, just like the Product ID. Though this is typically not necessary or appropriate, it does allow you to create special builds of your application that will only activate with a certain Product Option ID. Using the "XYZ Product" example here, this kind of validation can be useful if the "Standard Edition" and "Express Edition" are released and distributed as entirely separate builds/installers. Otherwise, if both editions are made available in a single build/installer, then it is best to rely on Product ID validation alone.

Automatic Recurring Payments

SOLO Server's Payment Plans allow you to offer subscription licenses, maintenance and support subscriptions, payment over multiple installments, and more. To learn more about how to configure SOLO Server for Payment Plans, please refer to the **SOLO Server manual**.

Creating SOLO Server Licenses

Important

Test licenses are available for **your** development and testing needs as you may deem necessary, but test licenses should not be distributed to customers. At the end of each calendar month, all test licenses are purged from SOLO Server. You will not be charged for creating test licenses.

All other non-test (or "live") licenses created in SOLO Server (regardless of how the licenses are created) will **not** be purged, however, and **will result in account charges/fees** being assessed to your SOLO Server account. There are various ways of creating non-test licenses, including, but certainly not limited to the following examples: Manually adding a license, processing an order without using the test credit card number, or performing Web service calls.

Adding and Activating Test Licenses

To test activation, you must first generate a license to activate. To do this, **sign-in to SOLO Server**, go to the menu *Customers / Add a Test License*. Find the appropriate product option in the list, and click the *Add New Test License* button. If you are evaluating SOLO Server, then select the "Protection PLUS 5 SDK Sample Full License" product option from the list. Now you can copy the License ID and Activation Password from SOLO Server and paste this information in for either of the activation screens.

If you get an error response during your activation attempt, review the **references** available and/or **contact us** for assistance.

Adding Live Licenses

Before you can add a live license, you must first locate or create a customer record for which the license will be issued. You can locate a customer using the *Search* menu, and you can add a customer using the *Customers / Add a New Customer* menu item. Once you have located or added the customer record, you should arrive at the *Customer Details* page. If not, you should find a small *View Customer* link in the bread crumbs, which are just under the main menu. Then, follow the steps below:

1. From the Customer Details page, scroll down to the *Licenses & Other Items* section, and click the *Add License* button.
2. Select the appropriate Product Option from the list.
3. From here, you may click the *Begin New Cart* button and use the shopping cart to process the order, or use the *Add New Prepaid License* button to add a license without using the shopping cart:

Begin New Cart

Using the shopping cart while logged into SOLO Server allows you to in put and charge a credit card for the order, or process it as a prepaid order (which is typically done if you have already received payment). Using the shopping cart will always automatically send an order confirmation email to the customer. Simply complete the checkout process to create the license, and the customer will be notified of their order.

Add New Prepaid License

If you do not want to send the customer an automated email notification, and/or do not need to process a payment, then use the *Add New Prepaid License* button. This will show an Add License page with default product option values that you can adjust if needed. If you are uncertain about any of the fields, click the *Help* icon at the top right of the page or use the menu *Help / Help for this Page* to see more information in the SOLO Server manual. Click the *Submit* button to add the license, and then provide the customer the License ID and Activation Password, which they need to be able to activate the software.

Working with Network Floating Licenses in SOLO Server

Network Floating Licensing is where an application may be restricted to running on a specific network, and restricted to a certain number of concurrent seats (where a seat may be a user or running instance of your application).

After adding a license in SOLO Server, you can edit the license details to customize additional settings. When editing these details, you can change the number of network seats a given license may allocate by modifying the *License Counter* value. When using semaphores for network floating, the license will need to be refreshed after applying this change.

Deactivating Licenses Remotely with SOLO Server

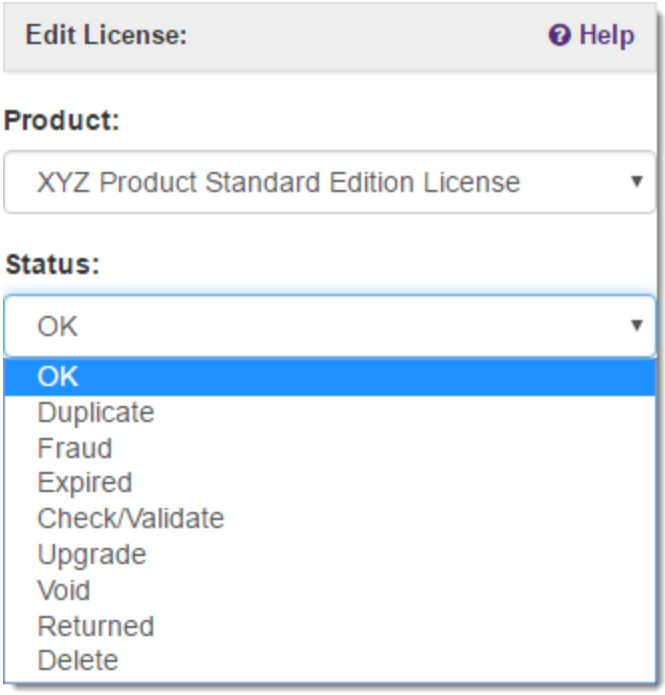
One of the core **ELM features** of SOLO Server is the ability to remotely deactivate licenses. This relies on background checking and refreshing via Protection PLUS 5 SDK in your application, which is needed to periodically check with SOLO Server to see if your application's license has been deactivated. There are two different ways in which a license may be deactivated: you can deactivate an entire license, which means any system previously activated will no longer be authorized to use your protected application; or you can deactivate an individual system that has been activated.

Deactivating an Entire License

Deactivating an entire license in SOLO Server allows you to disable any system which was activated with a particular license. This can be useful in scenarios such as:

- A user purchases a license for your application and activates it, but later requests a refund.
- Similar to the above, a user purchases your application, but does so with fraudulent payment information, which could result in a charge-back.

To deactivate a license, simply **sign-in to SOLO Server's administrative interface**, find the appropriate License ID and bring up the License Details page (you can get to this page by searching for the License ID, or by searching for the customer or invoice and clicking on the License ID's hyperlink shown below the customer or invoice details). Then, click the Edit link on the License Details page, and change the status to something other than 'OK' (as shown below).



In most cases, you would change a license to Duplicate if the customer purchased a second license by mistake, Fraud if the payment has been disputed, Expired if there is some other general reason to disable the license, or Check/Validate if there is a reason that the license should be reviewed. The Upgrade status will only deactivate the license if the Upgrade Behavior is set to Deactivate in the product option settings for the license.

For self hosted SOLO Server users with the Master User ID permission, the status can also be set to Void if there is some other general reason to disable the license, or Returned if the customer requested a refund.

Deactivating an Individual System

When a customer activates a license using SOLO Server, he or she is issued an "Installation ID" which corresponds with that activation. Since each activation receives its own unique Installation ID, it is possible to manage and deactivate individual computer's or systems which were previously activated. An example of when you would use this functionality would be when a customer encounters a hardware/computer failure that renders the prior activation unusable. By deactivating the Installation ID associated with that computer, you can rest easy that your protected application will see that it has been deactivated the next time it does a background check or refreshes its license. Additionally, deactivating an Installation ID will increment the number of activations allowed for a license, so doing this also enables your customer to activate on a new computer.

An Installation ID can be deactivated remotely through the SOLO Server author interface. **Sign-in to SOLO Server's administrative interface**, find the appropriate License ID and bring up the License Details page (you can get to this page by searching for the License ID, or by searching for the customer or invoice and clicking on the License ID's hyperlink shown below the customer or invoice details). Note that the "Deactivations Left" (on the License Details page) will need to be 1 or greater for deactivation to be available. Then, click on the "View Installation History" link at the top, find and click on the Installation ID that corresponds with the computer which was originally activated, and click the Deactivate Installation button. You can also search by Installation ID directly, but if the "Deactivate Installation" button is not present, you may need to click the "View License" link in the bread crumbs (near the top of the page) and increment the "Deactivations Left" field to enable this option.

An Installation ID can also be **deactivated remotely though the customer license portal** by the customer if you choose to allow this option.

SOLO Server Result Codes

A list of result/error codes which may be returned by SOLO Server web services is below. The most current version of this list is **available online**.

Number	Summary	Details
5000	**Invalid EncryptionKeyID**	For security purposes, these Web services use public-private key encryption and digital signatures. In order to determine the keys used on the server side for decryption/encryption and signature verification/creation, each request is required to include an EncryptionKeyID XML element containing the author's unique EncryptionKeyID. This Return Code is generated when this element is either missing or the EncryptionKeyID itself is not valid.
5001	**Encryption Required**	For security purposes, these Web services require that all XML content aside from the EncryptionKeyID be encrypted and include a digital signature of the encrypted portion by the keys corresponding to the EncryptionKeyID. This Return Code is generated when this portion of the XML is not properly encrypted.
5002	**Signature Required**	For security purposes, these Web services require that all XML content aside from the EncryptionKeyID be encrypted and include a digital signature of the encrypted portion by the keys corresponding to the EncryptionKeyID. This Return Code is generated when digital signature cannot be verified.
5003	**Decryption Failure**	For security purposes, these Web services require that all XML content aside from the EncryptionKeyID be encrypted and include a digital signature of the encrypted portion by the keys corresponding to the EncryptionKeyID. This Return Code is generated when the encrypted portion of the XML cannot be successfully decrypted.
5004	**Verification Failure**	For security purposes, these Web services require that all XML content aside from the EncryptionKeyID be encrypted and include a digital signature of the encrypted portion by the keys corresponding to the EncryptionKeyID. This Return Code is generated when digital signature cannot be verified.
5005	**Invalid Parameters**	Each Web method has optional and required input content. This Return Code is generated when one or more required input elements are missing.
5006	**Security Check Failure**	Too many failed activation attempts were made in a short period of time, suggesting the possibility of a "brute force attack" to acquire a successful activation. On the License Details page, click the "Reset Activation Check" button to clear the temporary block.

5007	**Invalid ComputerID**	The UserCode2 (ComputerID) value in the request XML is either an invalid numeric value or 0 (zero).
5008	**Invalid Activation Data**	The License ID and/or Password could not be validated by SOLO Server. Make sure the user has a valid License ID and Password and is keying them correctly when attempting activation.
5009	**Unregistered Customer**	If the RequireRegistration XML element is included and specified as True for an activation request, the system will verify that the customer is registered and reject the activation if the customer is indeed not registered. This Return Code is generated when this condition occurs.
5010	**Invalid ProductID**	If the ProductID XML element is included and populated for an activation request, the system will verify that this value matches the ProductID associated with the License ID. This Return Code is generated when this check fails.
5011	**Invalid ProdOptionID**	If the ProdOption XML element is included and populated for an activation request, the system will verify that this value matches the ProdOption associated with the License ID. This Return Code is generated when this check fails.
5012	**Invalid Product Version**	The version of the application is less than the Minimum Activation Version specified on the Product Option.
5013	**Invalid Option Type**	The user has consumed the maximum number of activations allocated to the License ID.
5014	**Invalid Option Type**	An invalid Product Option Type has been specified for the associated License ID. You must select one of the PLUS Option Types to be able to use the license for activation.
5015	**Invalid InstallationID**	The InstallationID in the request XML is invalid.
5016	**Deactivated Installation**	The Installation ID has been deactivated within SOLO Server. A SOLO Server user must re-enable the Installation ID or issue a new License ID / Password to be able to continue.
5017	**Invalid License Status**	The License ID status, as displayed on the License Details page within SOLO Server, is something other than "OK".
5018	**No Remaining Deactivations**	Deactivation failed due to the absence of remaining deactivations for the License ID.
5019	**License Expired**	The specified expiration date of the License ID is now in the past.

5020	**Internal Authentication Failed**	Authentication failed for an internal call to a Web method.
5021	**Invalid IP Address**	The IP address of the activating machine is not among the Allowed Activation IPs as displayed on the License Details page in SOLO Server. This Return Code is only applicable if the Product Option associated with the License ID has been configured to restrict activation to one or more client IP addresses.
5022	**Invalid System Time**	The date/time as displayed by the system clock is not within the threshold specified by SOLO Server. The user must correct the system clock to be able to continue.
5023	**PLUSNative Access Not Enabled**	A call to a web service method which requires access to SoftwareKey's PLUSNative fails because either the author does not have a PLUSNative license or the license has yet to be enabled by SoftwareKey.
5024	**Invalid XML Document**	The XML document is not properly formatted and cannot be loaded.
5025	**Plug-in Failed**	When a pre-activation plug-in is configured on a Product Option, this Return Code is generated during activation when there is an error loading or executing the plug-in not handled by the plug-in itself.
5026	**Plug-in Processing Failed**	When a pre-activation plug-in is configured on a Product Option, this Return Code is generated during activation when the plug-in processing indicates that the activation should be rejected.
5027	**Invalid Network Session**	The Network Session has been closed or has expired.
5028	**No Network Seats Remaining**	No more network seats are currently free. This error will continue to be returned for the LicenseID until another network session is closed or expires.
5029	**Invalid Trigger Code Data**	Invalid Trigger Code data was provided during activation.
5030	**Protection PLUS 5 SDK evaluation expired**	The Protection PLUS 5 SDK evaluation has expired.
5031	**SOLO Server evaluation expired**	The SOLO Server evaluation has expired.
5032	**Network Floating**	The SOLO Server Network Floating capabilities have not been enabled for

Access Not Enabled	the author.

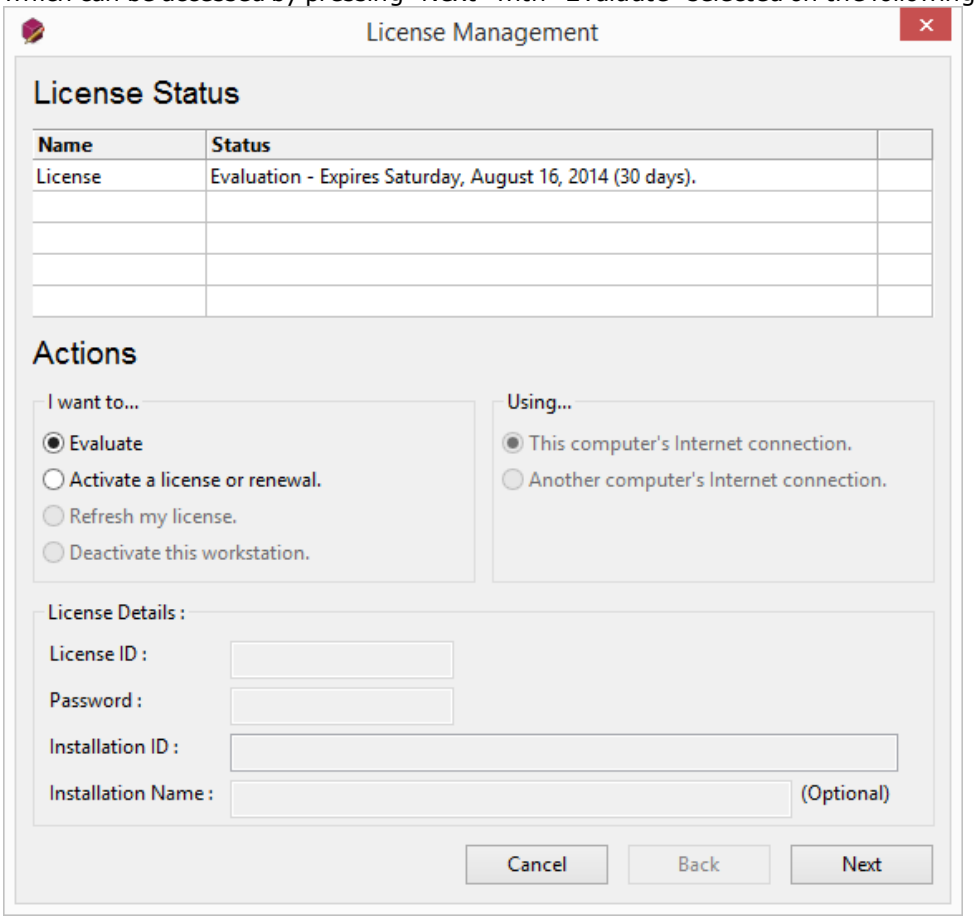
License Manager Overview

Most of our customers leverage the powerful automation capabilities of our central licensing server/service, SOLO Server, which includes **Electronic License Activation and Management** features such as:

- Online (connected) and offline (disconnected) users license activation 24 hours a day
- Periodic background license validation
- License updates / refreshes
- License revokation / deactivation
- Unsupervised license transfers
- Optional eCommerce features
- See the **Top 10 Features**

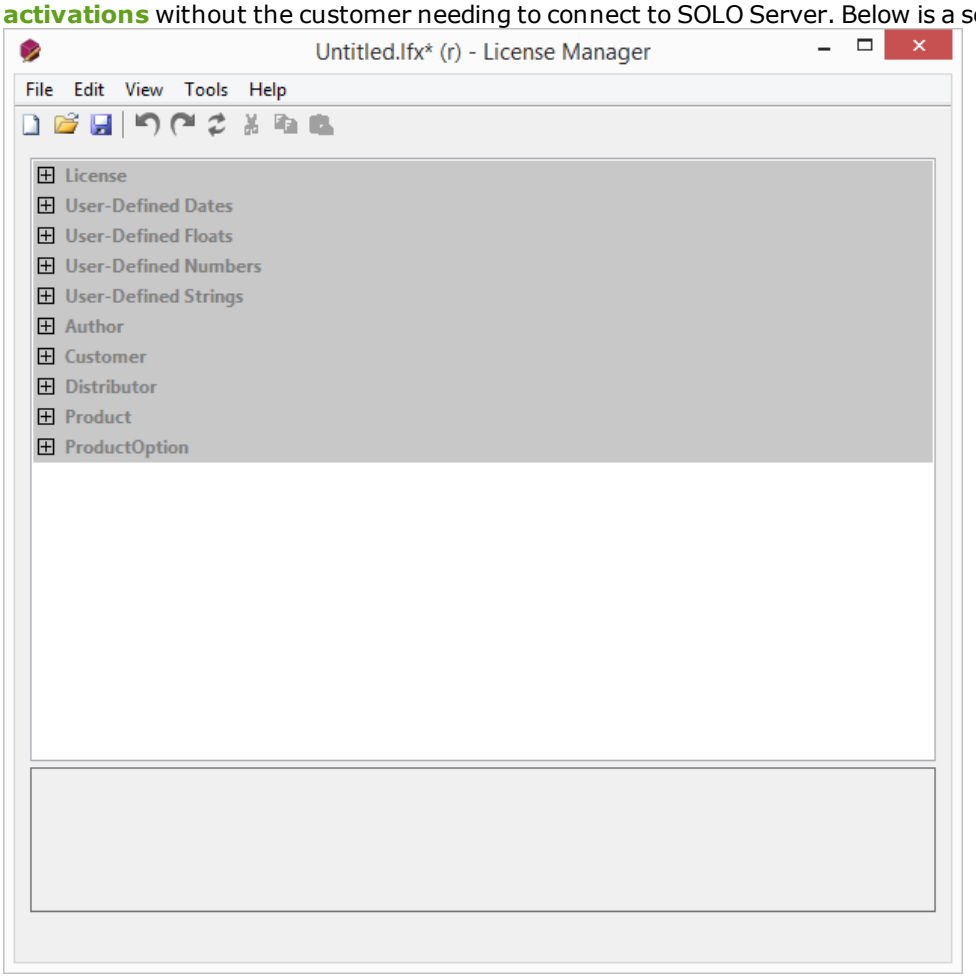
For those customers wanting to process everything manually, License Manager is a desktop GUI application allowing you to activate, open, view, create, edit, and save Protection PLUS 5 SDK license files without requiring SOLO Server.

If you haven't activated with a Protection PLUS 5 SDK license, License Manager will start up as a 30 day evaluation, which can be accessed by pressing "Next" with "Evaluate" selected on the following License Management dialog.



Getting Started

License Manager is similar to LFEdit from Protection PLUS 4, but presently does not have an equivalent to product definitions. Along with editing **license files**, it will let you **process manual activation requests** or **trigger code activations** without the customer needing to connect to SOLO Server. Below is a screen-shot of License Manager.



The toolbar buttons from left to right are New, Open, Save, Undo, Redo, Cut, Copy, Paste. You can always hover over the buttons to get a tool-tip if you are ever unsure of what they do.

Encryption Envelopes

Purpose

Protection PLUS 5 SDK **license files** are XML documents that have been encrypted and digitally signed to **secure** the contents and prevent unauthorized modification. The envelope contains the encryption keys needed to handle license file encryption and decryption. The envelope format helps secure these keys, and it is important for you to make a reasonable effort to secure/hide the envelope in your application so it cannot easily be found by examining the application or its memory.

Read-only **license files** are files which must be encrypted and signed by License Manager or SOLO Server. The advantage to these license files is that they are digitally signed with key data that is only partially known to your application. The drawback is that when using License Manager, you must manually generate the license file and there is no tracking in a database like there is with SOLO Server. To use SOLO Server, the drawback is that you will need Internet connectivity, but your protected application will not necessarily require Internet connectivity.

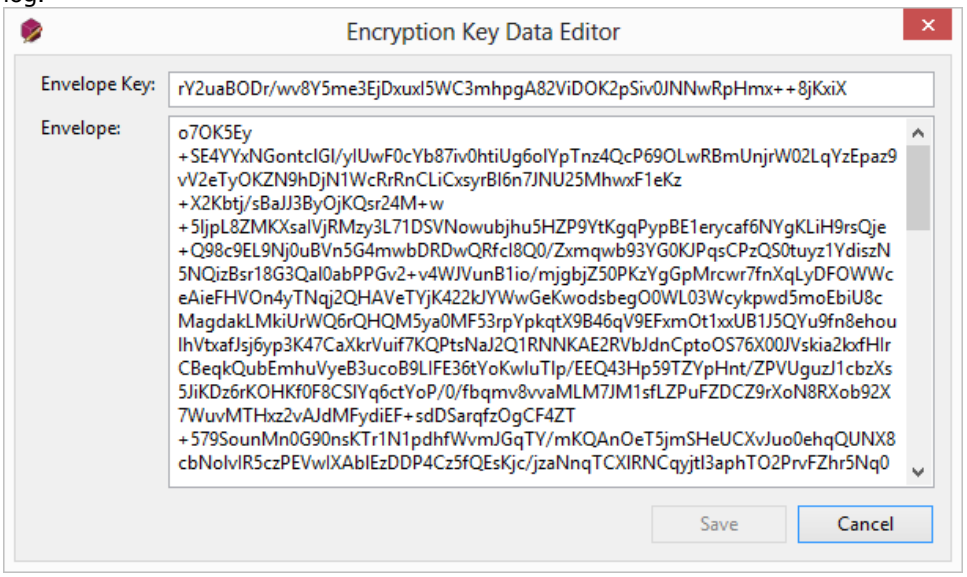
License Manager will also work with writable **license files**, which are license files that are encrypted and signed by your application.

License Manager Envelope VS. Protected Application Envelope

License Manager's envelope has private key data for the server key, which is needed to create read-only **license files**. If you distribute this key with your application, you basically make all license files writable, which is less **secure**.

The protected application envelope will be added into your application's source code.

The `AuthorEncryptionKey` (usually configured in the `LicenseConfiguration.EncryptionKey` property in the Protection PLUS 5 SDK sample projects) contains the sample Envelope Key and Envelope for protected applications. This corresponds with the default License Manager envelope key and envelope in License Manager's *Tools / Options* dialog.



For your own applications, you will need to change these to your own envelope values, both in your source code and License Manager.

How to Get Your License Manager Envelope

Open a support ticket with us and make a request for "Protection PLUS 5 SDK Encryption Key Data". License Manager's envelope must **never** be shipped in or with your protected applications, and must only be used in License Manager. We recommend making a secure backup of your envelope data, as we do not keep our customers' encryption envelopes on file. We can generate a new encryption envelope if necessary, but it would not match up to the applications deployed with the other encryption envelope data.

If you are not using SOLO Server, we will also include the "Protected Application Encryption Key Data" at the same time. If you are using SOLO Server, we will generate the License Manager Envelope that matches your SOLO Server account.

Important

Envelopes for License Manager are not the same as envelopes distributed with protected applications! Do NOT include the License Manager envelope in or with your protected applications. In your protected applications, only use the protected application envelope, which we can send you through a support ticket or it can be generated by SOLO Server (*Configure / Products* then use the Actions dropdown and choose *View Encryption Key Data*).

How to Get Your Protected Application Envelope

If you are **using your own SOLO Server account**, you can get the envelope key and envelope by choosing the menu *Configure / Products* then use the Actions dropdown and choose *View Encryption Key Data*. If you are not using SOLO Server, **open a support ticket** with us and make a request for "Protection PLUS 5 SDK Encryption Key Data". This will include both the License Manager envelope (see above) and protected application envelope.

Managing License Files

Protection PLUS 5 SDK **license files** contain all the license data and parameters needed by your application. License Manager categorizes the data to make it easier to manage and edit. Each category has a set of fields, and the **license file schema** describes each field in detail. Before configuring the license fields, it is important that you have an **understanding of licensing** and have **defined your licensing requirements**.

The field descriptions in this topic are also displayed at the bottom of License Manager's window for each category and field.

Categories

Category	Description
License	In SOLO Server, each license record is created from a Product Option, and always belongs to a customer record. Without SOLO Server, each license file is created by License Manager without a corresponding license record.
User-Defined Dates	User-defined date fields 1 through 5, which may be used for arbitrary dates when needed for additional, customized license data.
User-Defined Floats	User-defined float fields 1 through 5, which may be used for arbitrary decimal values when needed for additional, customized license data.
User-Defined Numbers	User-defined number fields 1 through 5, which may be used for arbitrary integer values when needed for additional, customized license data.
User-Defined Strings	User-defined string fields 1 through 5, which may be used for arbitrary alpha-numeric values when needed for additional, customized license data.
Author	You or your company is the "Author" of the software or application being licensed.
Customer	The Customer represents your customer, for whom one or more licenses have been issued.
Distributor	In SOLO Server, a distributor represents a company or individual that resells your products or applications, or an affiliate which refers customers and prospects to your web site. The distributor information can be particularly useful in scenarios when the distributor acts like a reseller, and also provides its customers with technical support.
Product	Each Product defined typically represents the Product or application being licensed (i.e. "XYZ Product"). Products must contain one or more Product Options, as all licenses are created from Product Options.
ProductOption	Product Options define licensing or purchasing options available under a Product. These can reflect any number of unique licensing and/or purchasing options, for example: "1 Year Subscription", "1 Year Subscription with Backup CD", or "1 Year Subscription Renewal".

Fields of Primary Interest

The following fields are important when you are using License Manager, and most licensing implementations will need data in these fields. If you are using the **Protection PLUS 5 SDK samples**, it will be expecting data in some of these fields (see below).

Field	Description
LicenseID	A unique, numeric identifier for the license issued (typically issued by SOLO Server). If you are using License Manager without SOLO Server, you will want to keep track of what License ID you issue to each customer.
ProductID	A unique, numeric identifier used to identify the Product for which a license was issued. This value is typically generated by SOLO Server. If you are only using License Manager, you can choose your own value for the Product ID. You can choose to verify this value in your source code.
ProdOptionID	A unique, numeric identifier used to identify the Product Option for which a license was issued. This value is typically generated by SOLO Server. If you are only using License Manager, you can choose your own value for the ProductOptionID.
SignatureDate	The date in which the license file itself was created and signed by License Manager or SOLO Server. This is automatically set when you save the license file with License Manager.
EffectiveStartDate	The date in which the license is effective. This is typically the date in which the license was created in License Manager or added in SOLO Server.
EffectiveEndDate	This typically reflects the date in which the license expires (for time-limited licenses), but it is also possible for your application to use this date for other purposes when the license is not time-limited, such as notifying customers when support/maintenance subscription periods are about to expire. When using SOLO Server, this field corresponds with the "Download Until" date of the license record.
LastUpdated	Only available when using writable **license files**, this reflects the last date and time in which the application was run. This is automatically set when you save the license file with License Manager.
InstallationID	When using SOLO Server, an Installation ID is created when the application is activated so that **Electronic License Management (ELM)** may be leveraged on the individual system. Activation done through SOLO Server uses a License ID and Password, which are not necessarily required when using License Manager. An Installation ID will not be auto-generated by License Manager
InstallationName	An optional, human-readable description of an Installation ID. This is typically specified by the user to help describe which of several systems he or she owns is being activated (a user could enter something like "Home Desktop" or "Work

	Laptop").
ActivationData	Contains the system identifiers used to identify the system which is authorized to use the license file.
TriggerCode	The Trigger Code number is a numeric value between 1 and 50, and is typically used to reflect the type of license being issued (i.e. non-expiring, time-limited, etc...). This value is configured manually in License Manager or in a Product Option in SOLO Server, and can be passed during a trigger code activation.
TriggerCodeFixedValue	The Trigger Code Fixed Value is a numeric value up to 14 bits in size (or a maximum value of 16383), which is typically used to conditionally enable application features, or distinguish between different product or application editions (i.e. "Express Edition" and "Enterprise Edition", etc...). This value is configured manually in License Manager or in a Product Option in SOLO Server, and can be passed during a trigger code activation.

You can use the `LicenseCustomData` field or the `User-Defined*` fields to store whatever custom data or parameters you want. This data will be available to your licensed application just like any other field.

Configuring the license fields

When you are updating a license file or **processing manual activation requests**, you can edit the values of the license file that your customer will receive. Certain values in the license file can be validated against values in the source code. The license file can also contain other data or parameters needed by your application. The following list will detail which license file fields need to be configured in License Manager to activate or run the **Protection PLUS 5 SDK samples:**

Field	Value	Description
License ID	any positive integer	If you are not using SOLO Server, this can be any positive integer and should be unique for each license.
Product ID	212488	The sample projects use the default value of 212488 for the ProductID which it uses to validate against the license file.
Signature Date	any date in the format: YYYY-MM-DDTHH:MM.SSZ	This is set automatically when you save the license file with License Manager. This date will be before the customer's system date if they haven't altered their system clock.
Installation ID	any non-empty string	When using SOLO Server, an Installation ID is created when the application is activated so that **Electronic License Management (ELM)** may be leveraged on the individual system. Activation done through SOLO Server uses a License ID and Password, which are not necessarily required when using License Manager. An Installation ID will not be auto-generated by License Manager, but the samples expect any non-empty string.

You can configure different fields, including the `TriggerCode` field, the `User-Defined*` fields, or the `LicenseCustomData` fields to pass custom information for your application to utilize. This data is typically used to determine things such as the type of license being issued (e.g. perpetual/non-expiring or time-limited), the version/edition of your application being licensed (e.g. "Express Edition" or "Enterprise Edition"), or details about specific features or modules that should be enabled or disabled. Other fields, such as `EffectiveEndDate` and `LicenseCounter`, can be used to restrict the use of you application based on date, number of users, etc.

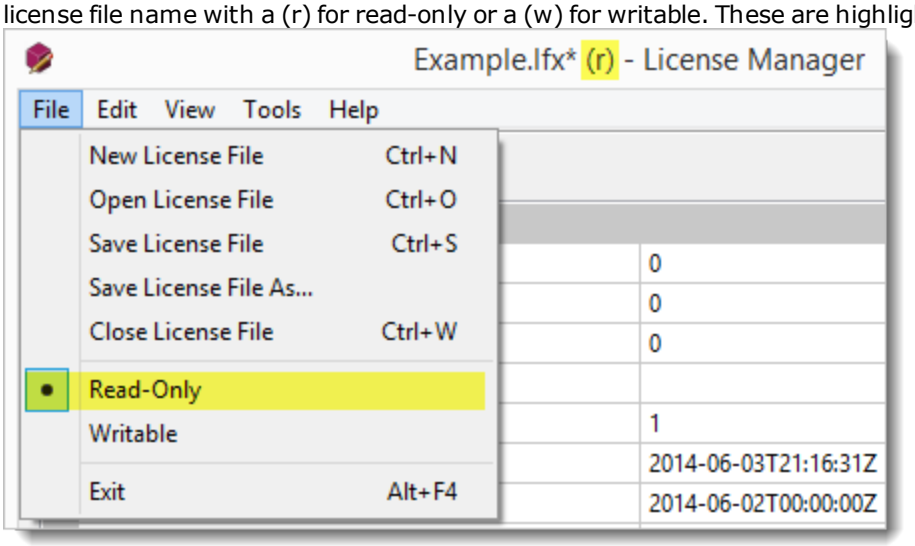
Field	Value	Result
Trigger Code	1 - 50	Your application can set the license type based on the Trigger Code it reads from the license file. In our **sample projects**, a Trigger Code value of 11 designates a time-limited license type, but you are free to use the Trigger Code values however you want. The Trigger Code is sent during a **trigger code activation**.
TriggerCodeFixedValue	int	This integer value can be read as bits enable features, or possibly designate which edition of the software should be activated. The TriggerCodeFixedValue is sent during a **trigger code activation**.
EffectiveStartDate	any date in the format: YYYY-MM-DDTHH:MM.SSZ	For a time-limited license, this field sets the beginning of the license period
EffectiveEndDate	any date in the format: YYYY-MM-DDTHH:MM.SSZ	For a time-limited license, this field sets the expiration of the license period
LicenseCustomData	any string	Custom string or XML formatted data specific to the license. Note that using long strings (anything over a KB or two) can cause performance degradation, and excessively long strings can also cause other problems.
User-Defined*	custom date, number or string	These fields can contain custody data. If also using SOLO Server, there is a 50 character limit on the size of the User Defined string/char fields. The `LicenseCustomData` field can potentially provide more flexibility, as it accepts XML formatted data.
LastUpdated	any date in the format: YYYY-MM-DDTHH:MM.SSZ	If using a writable **license file**, this field needs to be updated so that it is not overwritten on the user's machine by an alias file with a later date. This is set automatically when you save the license file with License Manager.

LicenseCounter	any positive integer	This will set the number of allowed network seats when using **network floating licensing**. It can also function as a usage counter that gets decremented or incremented.

Read-Only and Writable License Files

License Manager can create and edit read-only **license files** or writable **license files**. A read-only license file can only be generated by License Manager or SOLO Server, as it uses the envelope data not known by your licensed application. This provides an additional level of security. A writable license file can be created and updated by a licensed application.

To set the currently open license file read-only or writable, use the menu *File / Read-Only* or *File / Writable*. The menu will display a check or bullet to mark what is currently set. This is also reflected in the title bar next to the license file name with a (r) for read-only or a (w) for writable. These are highlighted in the following screen shot:



Processing Manual Activation Requests

Manual activation is the recommended method for activating an installation and obtaining a license file from License Manager, as this allows you to transfer all of the license file data. If there is no Internet connectivity or a way to transfer data or a file, using **trigger code activation** would be necessary, as this process can be done over the phone.

A manual activation request involves the end-user copying the request string or saving the request file that they would then send to you by email. You would paste the request string or load the request file into License Manager's manual request screen, which processes the request and generates a response as a response string or response file. The end-user must then take the response string or response file and load it into the protected application on the computer he or she is attempting to activate. The application can then parse the response and determine which actions to take.

To support manual activation requests in your application, you can view the topics on **manual activation with PLUSManaged** or **manual activation with PLUSNative**.

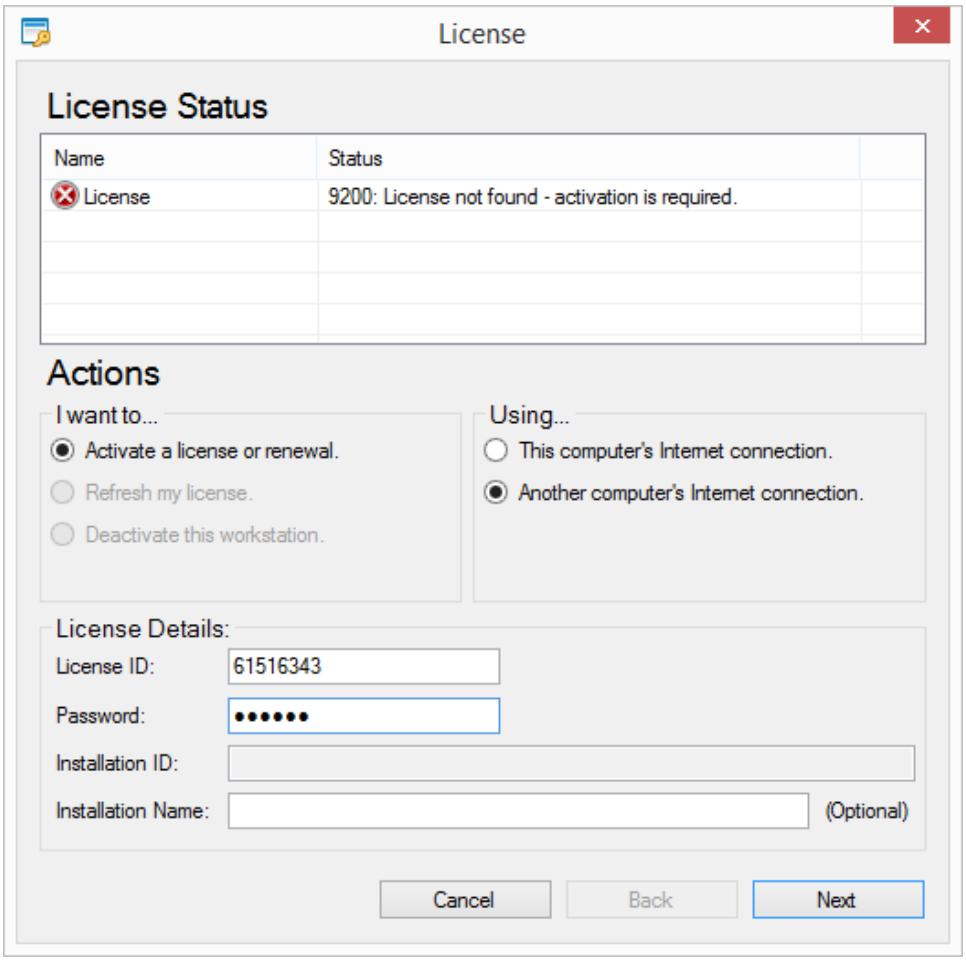
Important

If you are using SOLO Server to issue License IDs, this manual activation process should be done through SOLO Server instead of the License Manager.

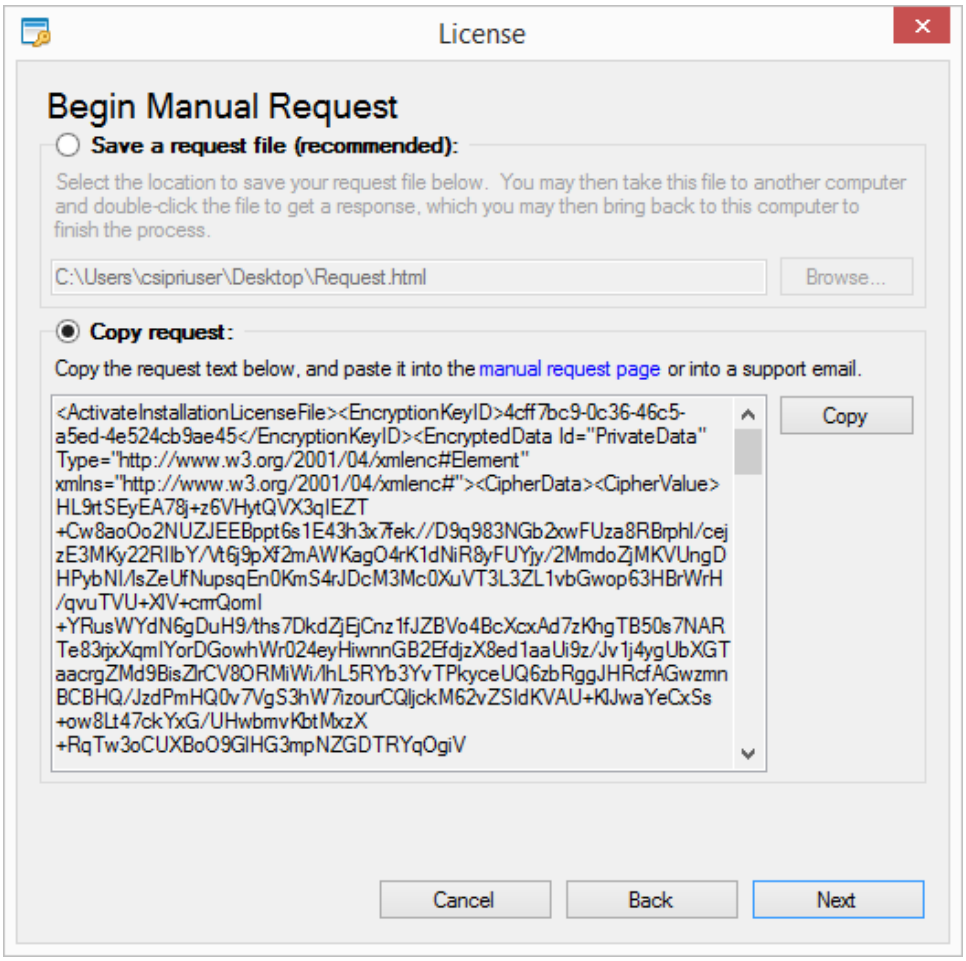
How to process a manual activation request with PLUSManagedGui samples

The following steps use the PLUSManagedGui read-only sample, but the same process applies to the other PLUSManagedGui samples. These steps will show both what your customer and you need to do to complete a manual activation with trigger codes. This process will also be very similar if you implement your own dialogs rather than using PLUSManagedGui.

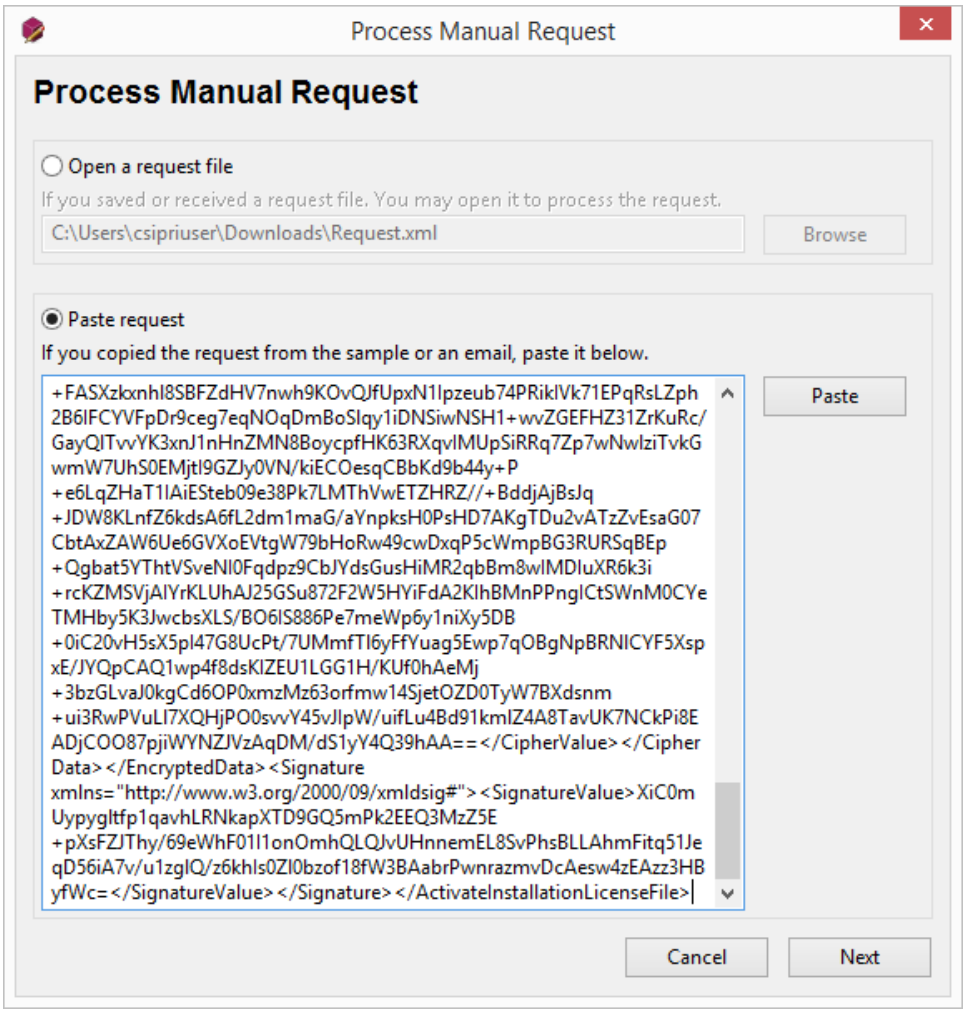
To start, your customer will see the following License Status screen, and they will need to select "Another computer's Internet connection". Note that this will be updated to clarify the new options with the License Manager. You can give the customer a License ID and Password to help you keep track of what type of license they purchased. With SOLO Server this is done automatically, but with License Manager you'll have to manually keep track of this. If the customer does not have a License ID and Password, they can enter any values (the password must be at least 4 characters) to get to the next step. We will use a sample License ID 61516343.



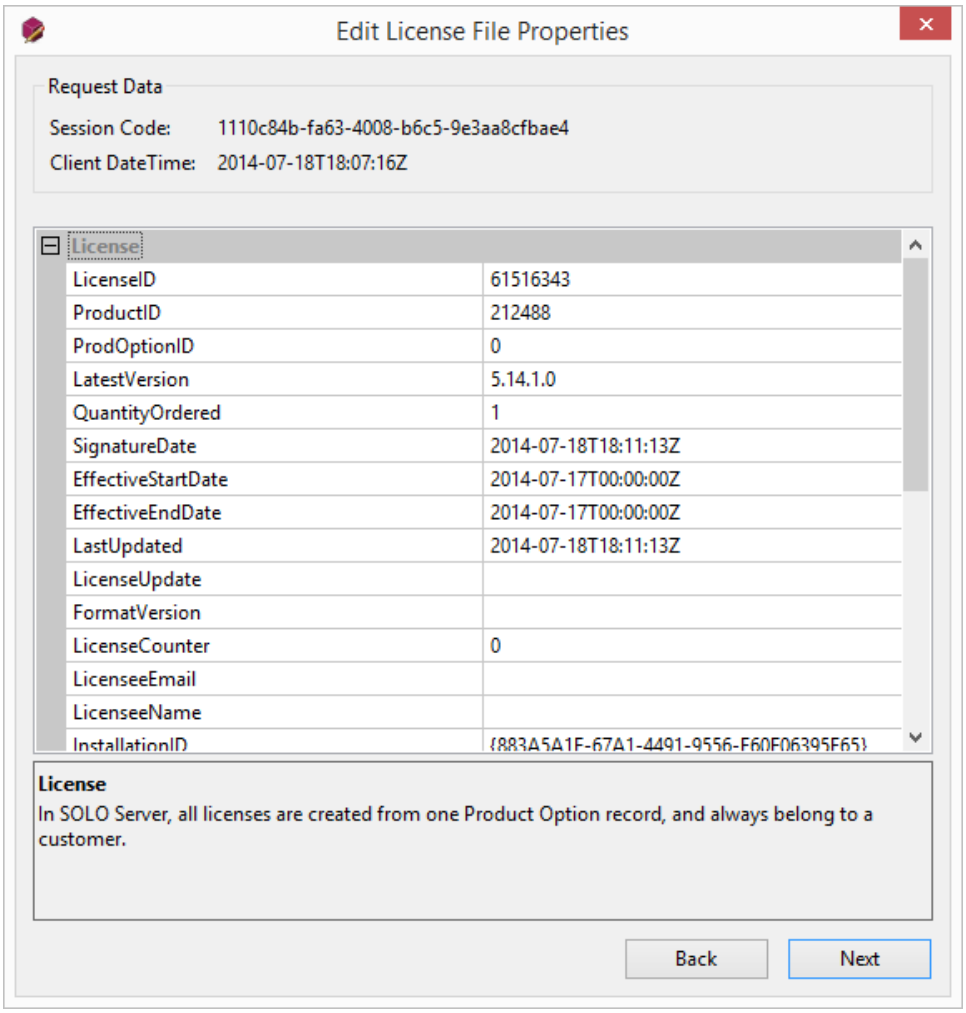
After the customer clicks "Next", it will bring them to the "Begin Manual Request" screen, where the request file can be saved, or the request can be copied to the clipboard (and later pasted), either of which must be sent to you by email. Our PLUSManagedGui create an html request file, which is meant to be used with SOLO Server. For this sample, you can copy the request text.



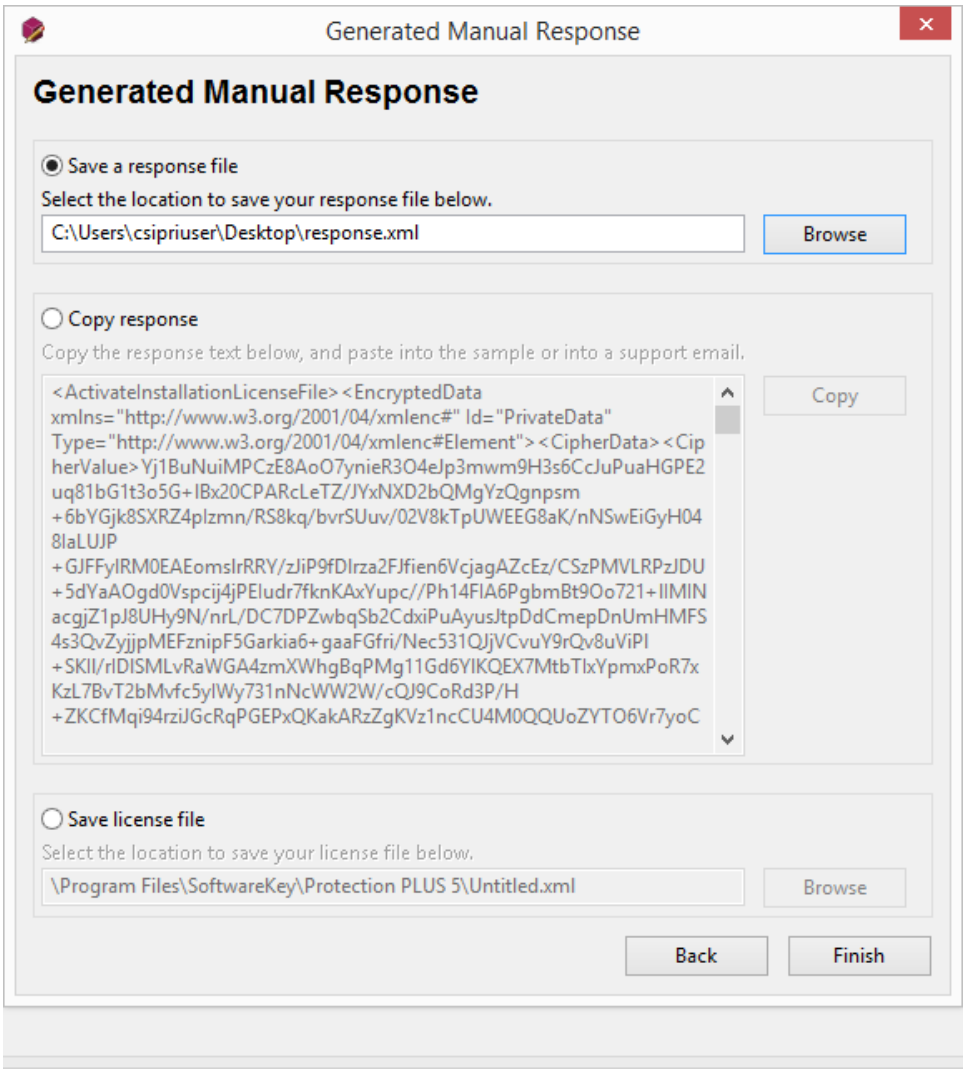
Once you have received the request file or the request text, you will open License Manager and choose menu *Tools / Process Manual Request* or press the F3 key. You will open the file or paste the text into the following screen.



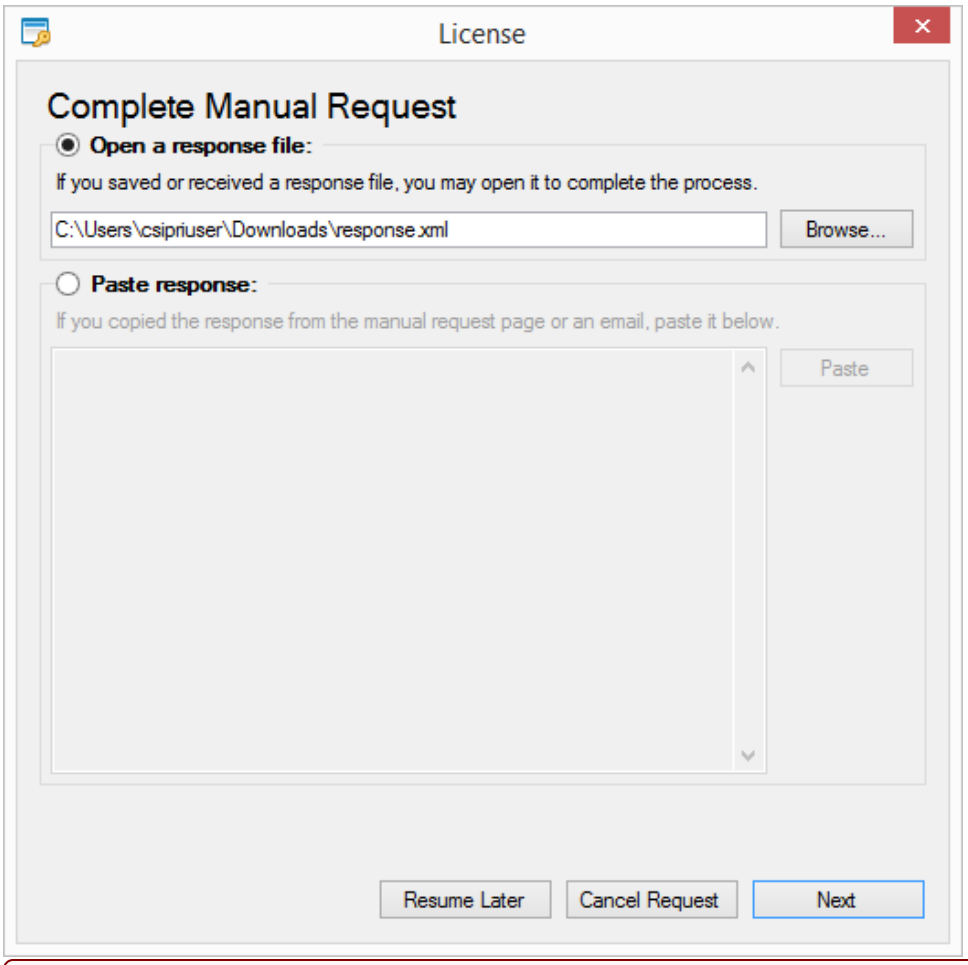
On the next step, you can edit the license file properties so that they apply to this specific customer and the license he or she purchased.



Once you've made all necessary changes to the license file and press next, License Manager will give you the generated manual response. You will now need to send this response back to the customer.



Back on the customer's machine, he or she will be waiting to complete the manual request. Once the response you generated has been received, your customer will be able to complete the activation.



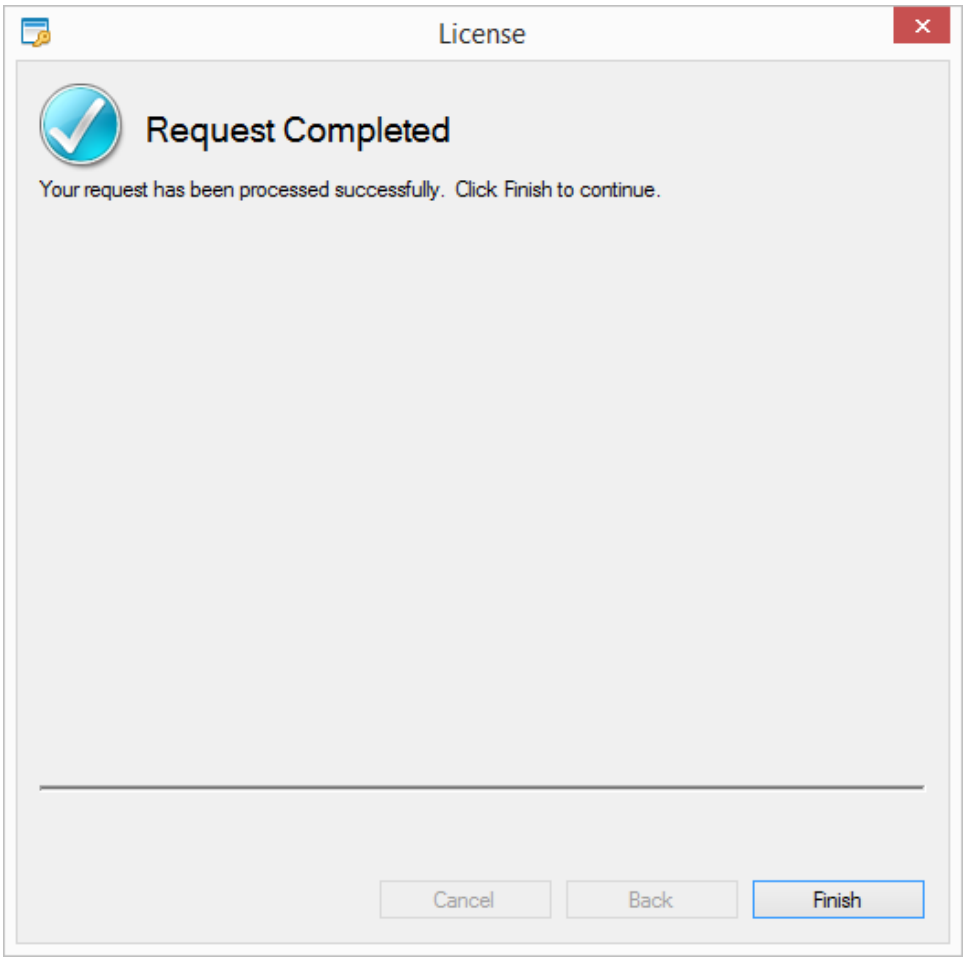
Important

If the user is pasting in a response instead of loading a response file, there are a few potential issues to be aware of:

- Users may be limited by the size of their clipboard if the response is too large. An"Invalid Data" error will occur if the entire response is not pasted in.
- For macOS users, if the encrypted response contains the characters "TM" and the user pastes in the response as a string, the operating system will automatically convert the "TM" characters to the trademark superscript character, which invalidates the response.

These issues do not occur when the encryption response is loaded from a file.

When successful, the request will be completed and the application will be activated.


License
Request Completed
Your request has been processed successfully. Click Finish to continue.
Cancel
Back
Finish

Processing Trigger Code Activations

Trigger code activations are primarily meant to be used for phone activations when no Internet connectivity is available. If Internet connectivity is available to you and your customer, even if only though a secondary device, we strongly recommend you use the **manual activation request**. If you are using SOLO Server to issue License IDs, this trigger code activation process should be done through SOLO Server to take advantage of its database and **electronic license management** (ELM) features.

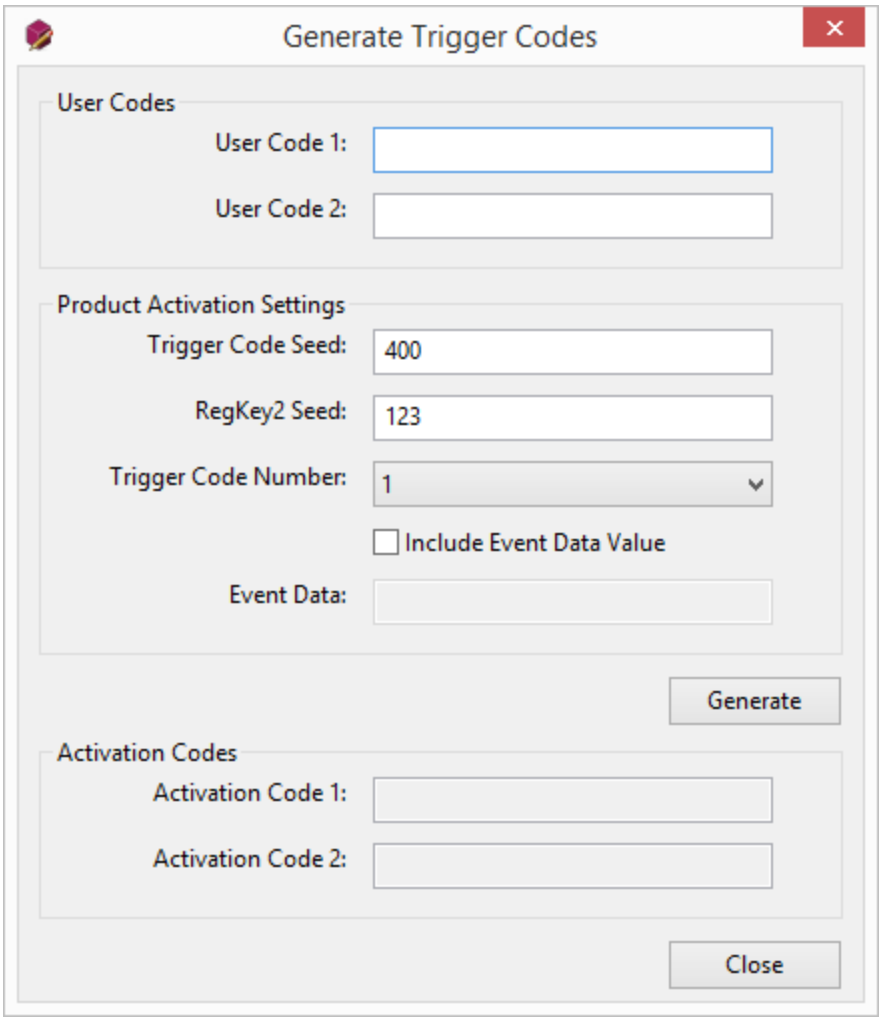
Review the information on **using trigger codes for telephone activation** for high-level details on how this process works. For more information on how to add support for trigger codes into your application, please view the manual topics for **trigger codes with PLUSManaged** or **trigger codes with PLUSNative.**

Important

Any challenge-response mechanisms such as trigger code validation can be reverse engineered (similar to how "key generators" are available on the Internet for many popular software applications). Therefore, it is best to use Protection PLUS 5 SDK's standard online and **manual activation** features when possible, which offer much stronger **security** than trigger codes.

Generate trigger codes with License Manager

To open the "Generate Trigger Codes" window in License Manager, go to the menu *Tools / Generate Trigger Codes* or press the F2 key.



Each of the fields and options are described below:

Fields / Options	Description
User Code 1	Also known as the "session code" to former Protection PLUS 4 users, this value is a randomized value that makes activation codes unique to each activation attempt. Its purpose is to prevent end-users from using an activation code more than once, which is very important when an activation code does something such as incrementing the number of network seats allowed. When users contact your company to activate, they will be required to provide this value.
User Code 2	Also known as the "Computer ID" to former Protection PLUS 4 users, this value is generated from device-specific information. Its purpose is to make each activation code unique to the PC for which it was issued, and prevent users from activating "Computer B" with an activation code generated for "Computer A". When users contact your company to activate, they will be required to provide

	this value.
Trigger Code Seed	This is the "seed" value used when generating Activation Code 1. Typically, this value should be unique between each of your products/applications so that you may prevent them from being cross-activated. (In other words, using a unique Trigger Code Seed value for each product prevents activation codes generated for "Product A" from being used to activate "Product B"). The value specified must be a number between 1 and 65535. When allowing activation through SOLO Server or License Manager, your application's source code and the corresponding product option (s) in SOLO Server or Product Activation Setting in License Manager must have matching values.
RegKey2 Seed	This is the "seed" value used when generating Activation Code 2. Typically, this value is also unique between each of your products/applications so that you may prevent them from being cross-activated. The value specified must be a number between 1 and 255. When allowing activation through SOLO Server or License Manager, your application's source code and the corresponding product option(s) in SOLO Server or Product Activation Setting in License Manager must have matching values.
Trigger Code Number	A Trigger Code Number is simply a number between 1 and 50, which defines the action(s) which will be taken when Activation Code 1 is successfully validated on your customer's computer. When a customer requests activation, you or your company are responsible for selecting the appropriate trigger code number when generating the activation code(s). (When using SOLO Server, this is configured on the product option selected when the license was created or purchased.) As the application developer, you are free to assign and program your own set of actions for any given Trigger Code Number. So for example, you could program your application to activate a non-expiring (or perpetual) license for your application when it receives a trigger code number of 1. At the same time, your application could also be programmed to activate a time-limited (or a "lease" or "periodic") license for some arbitrary number of days when it receives a trigger code number of 10.
Include Event Data Value	Checking this option will allow you to include an Event Data Value. This is necessary when you want do more than just activate a perpetual license.
Event Data	The specified value must be a valid integer between 0 and 16383 inclusive (14 bits in length). Based on the Trigger Code Number, you can have you application use this Event Data Value in different ways, such as setting an expiration date, enabling/disabling certain features, or setting the number of network seats. You are limited to a 14-bit value per trigger code activation.
Generate	This button will generate the Activation Codes once there are valid values for User Code 1, User Code 2, Trigger Code Seed and RegKey2 Seed.
Activation Code 1	When a user requests activation, this value is generated by you or your company. This value must always be provided to your end-users to complete an activation. Once your protected application has successfully validated Activation Code 1, it receives the Trigger Code Number, which allows your protected application to alter its license file as appropriate.
Activation Code 2	When a user requests activation, this value may (optionally) be generated by you or your company in order to send a secondary, numeric value to your application during activation. The

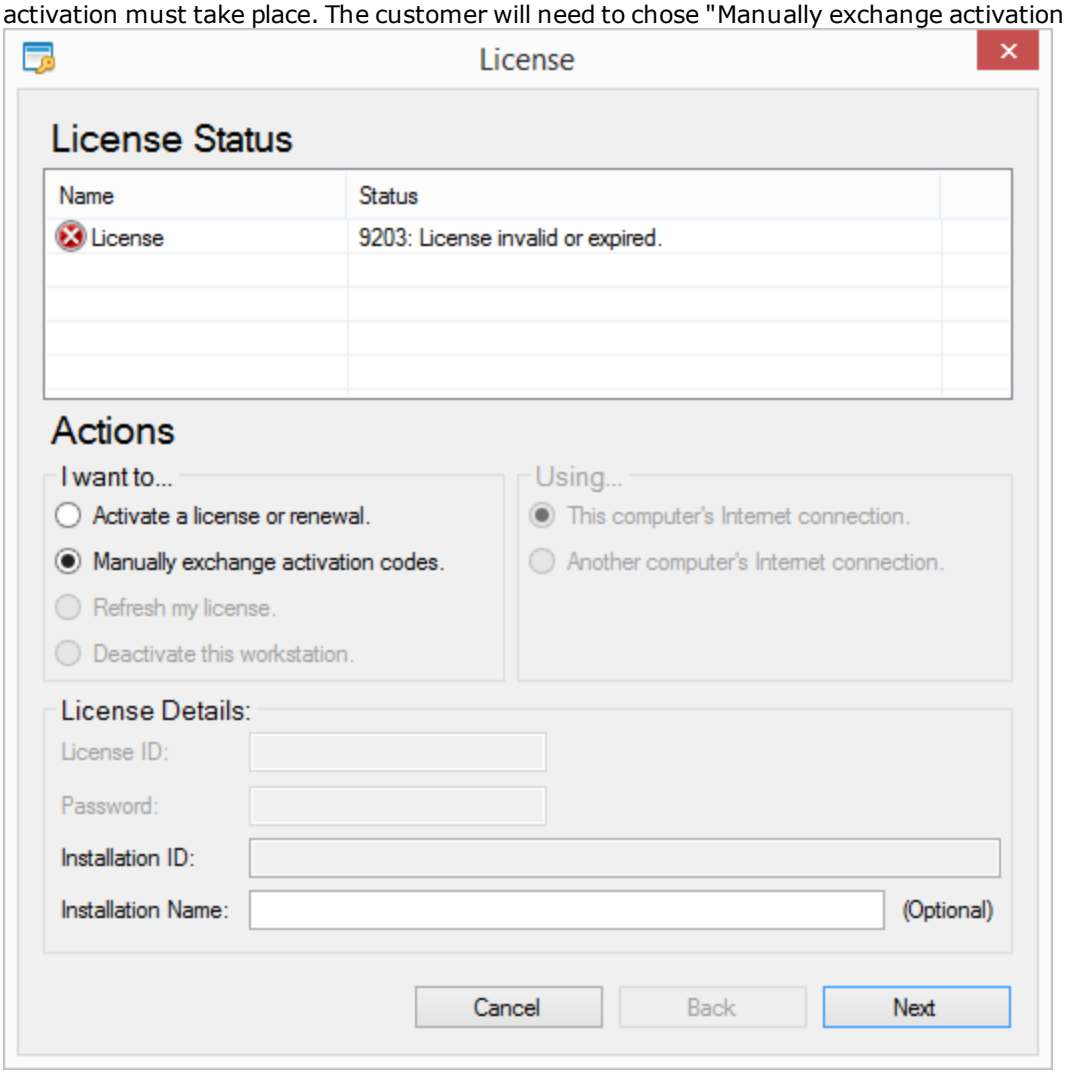
decoded value sent to your application through Activation Code 2 is the Event Data Value (see above).

The seed values need to be unique per application, or unique between non-free upgrade versions of your application. For previousProtection PLUS 4 users familiar with LFEdit, there is no product definition equivalent yet, so you need to manually document what seed values you are using at this point. The last seed values entered are saved for the next time the feature is used.

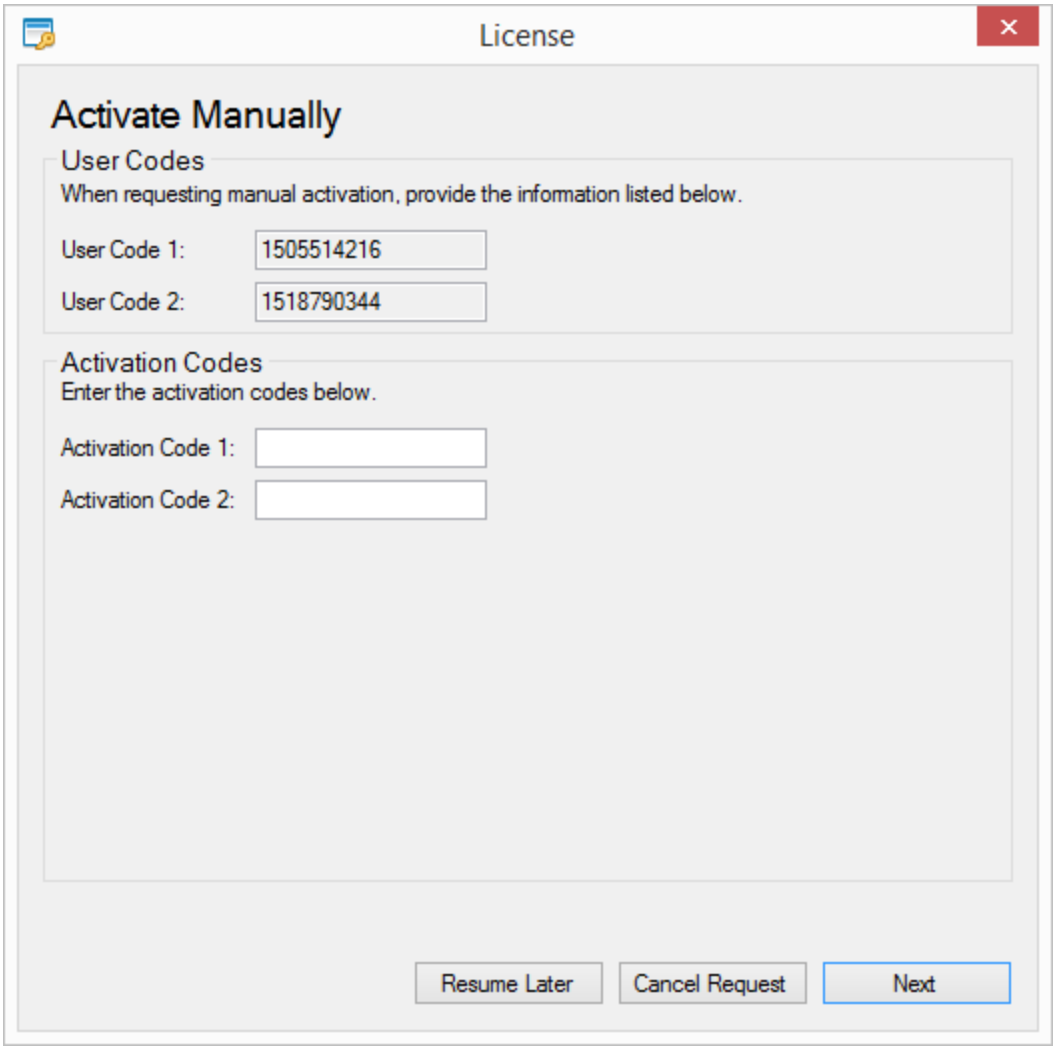
How to use trigger codes to activate a PLUSManagedGui sample

The following instructions use the PLUSManagedGui self-signed sample, but the process is similar in other PLUSManaged samples. This process will also be very similar if you implement your own dialogs rather than using PLUSManagedGui. These steps will show both what your customer and you need to do to complete a manual activation with trigger codes.

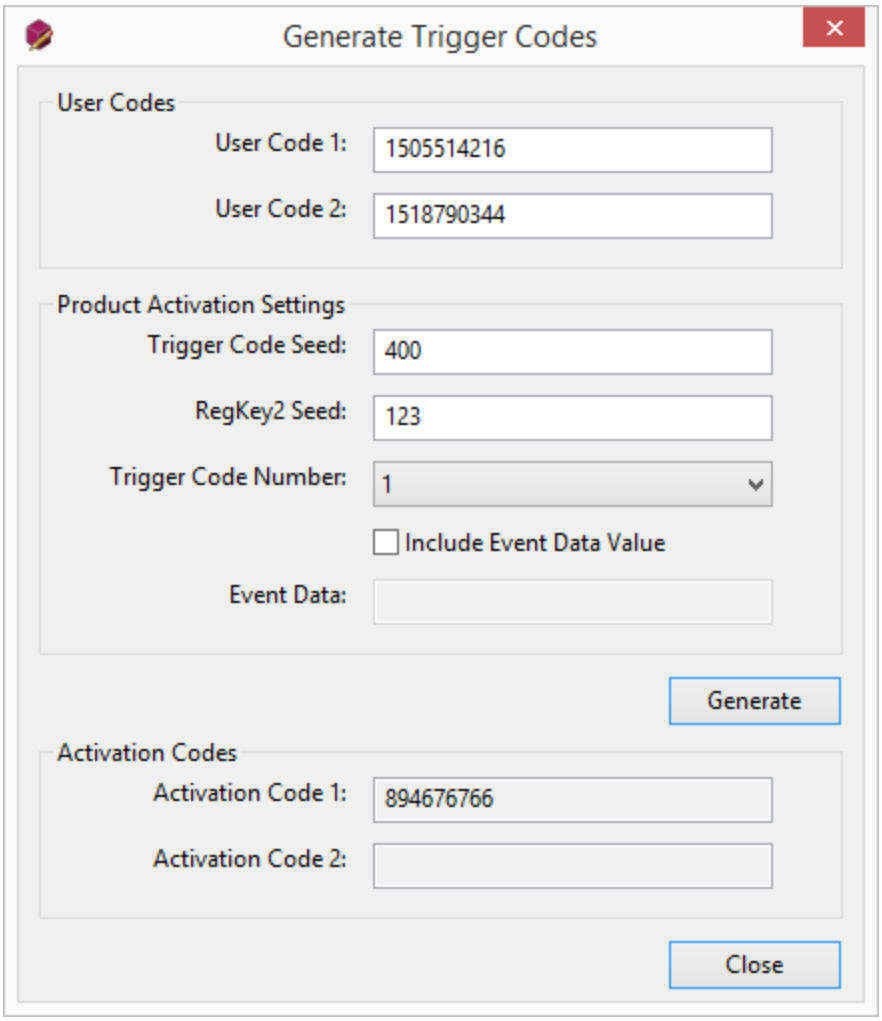
When the customer launches the application, PLUSManagedGui will show that no license file is found, and that an activation must take place. The customer will need to chose "Manually exchange activation codes" and click Next.



The customer will be given User Code 1 and User Code 2, which they need to communicate to you by phone, email, or text.

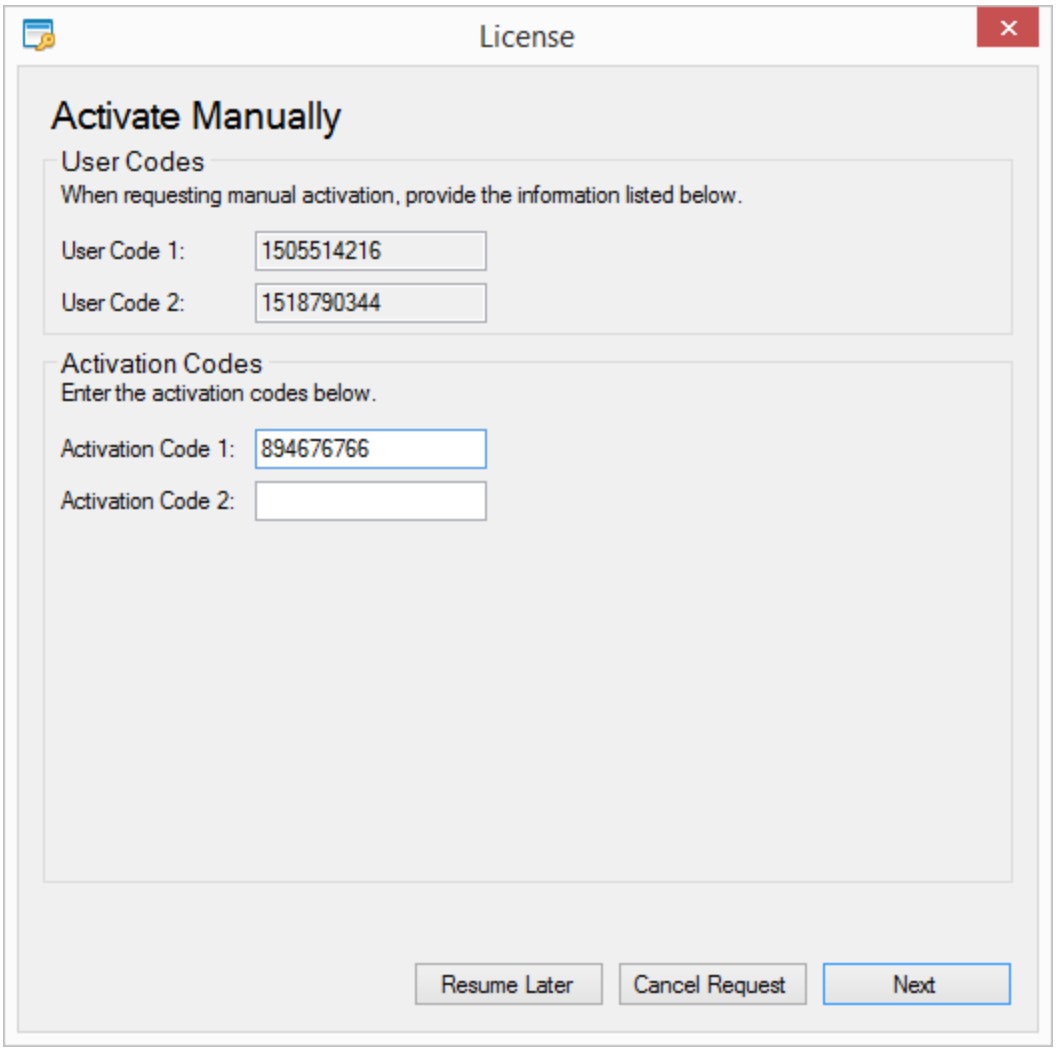


Now with License Manager, you will enter the User Code 1 and User Code 2 that the customer gave you. You will also enter the Trigger Code Seed and RegKey2 Seed values for your particular application, as well as the Trigger Code Number and Event Data depending on what type of license you are activating. Once you press Generate, this will give you Activation Code 1, and also Activation Code 2 if you are including the Event Data Value.

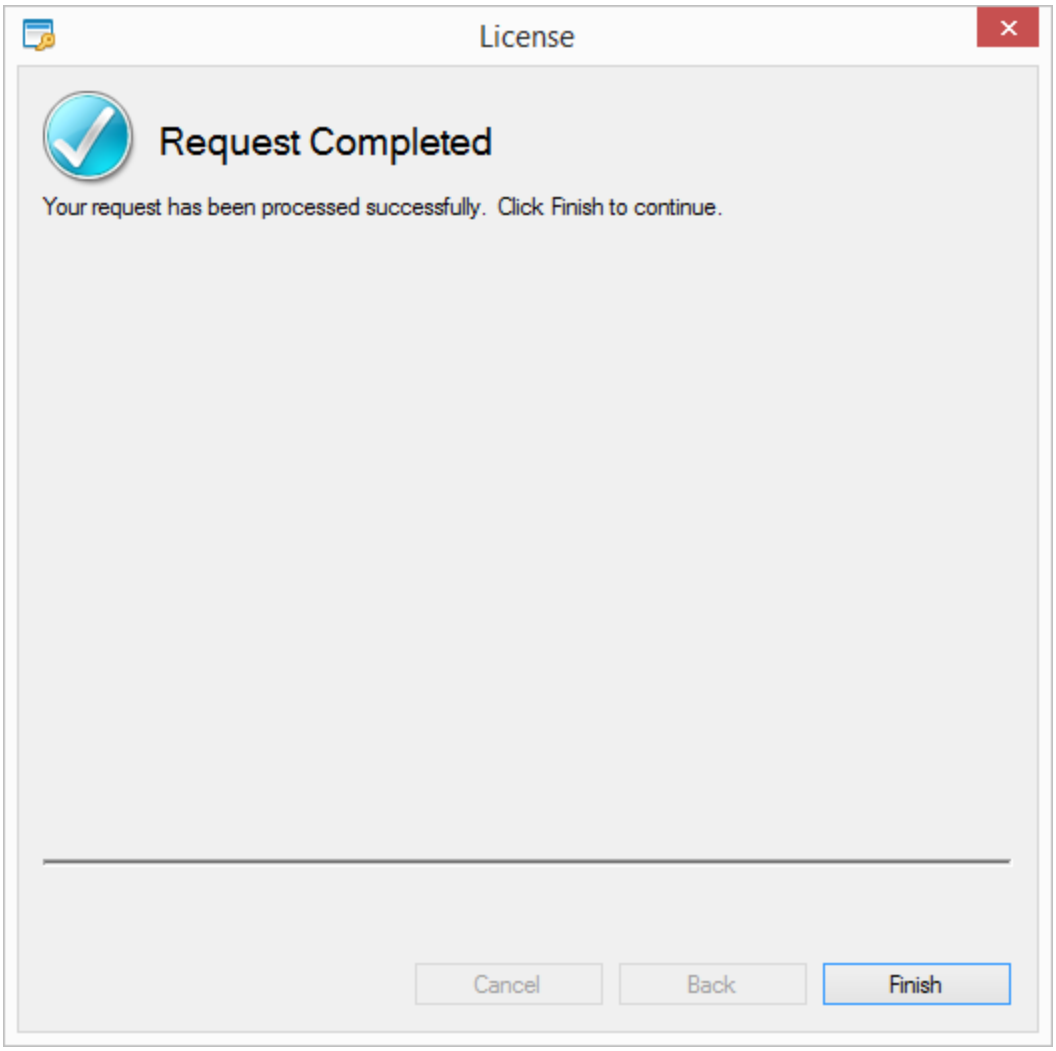


In all of our samples, the Trigger Code Seed default value is 400 and the RegKey2 Seed default value 123. These are set in the `LicenseConfiguration` class (in the Encryption Settings region), and you will want to configure these to your own values for your protected application(s). These **must** match the seed used in the license manger dialog for the activation to be successful.

You will now send Activation Code 1 (and Activation Code 2 if applicable) back to the customer by phone, email, or text. They will enter this into the activation screen and press Next.



If everything was done correctly, the request will be completed and the application will be activated accordingly.


License
Request Completed
Your request has been processed successfully. Click Finish to continue.
Cancel
Back
Finish

Customizing License Manager

The License Manager encryption key data and interface can be customized to help you better manage your license files. License Manager comes with a default profile, which includes two files located at *%PUBLIC%\Documents\SoftwareKey\Protection PLUS 5\License Manager* in Windows Vista and later. The first file is LicenseManagerProfile.xml, which initially contains sample encryption key data meant for evaluation and testing purposes only. **Never distribute production applications that use the sample/test envelope data!** Before releasing your applications, you must **request and use your own encryption key data**. The second file is the GridFieldLayout.xml file, which includes license file field/schema descriptions.

Creating a custom profile

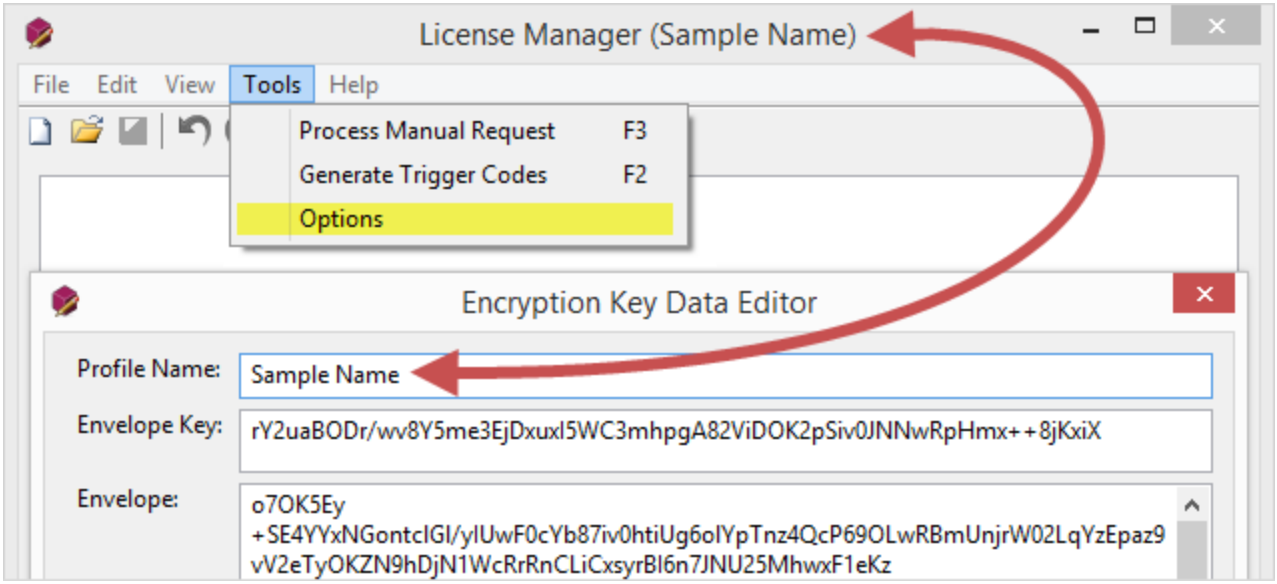
If you are managing products from different companies or departments, or if you have multiple SOLO Server author accounts (each of which will have different encryption key data), you would need to create and manage multiple License Manager profiles. You can also have a profile with customized license fields and descriptions. Before you begin configuring a new profile to use production encryption key data, you should first activate using your primary/default profile. Otherwise, you may need to activate each profile separately.

Setting up a new profile

1. Create a new directory/folder for your profile and copy the LicenseManagerProfile.xml and GriedFieldLayout.xml files into this new directory. For the purposes of this example, we will name this new directory so it matches the new example profile's name: *Sample Name*. We will create this folder under the default profile path: *%PUBLIC%\Documents\SoftwareKey\Protection PLUS 5\License Manager\Sample Name*.
2. Browse to where the License Manager shortcut is (this should be located in *%ALLUSERSPROFILE%\Microsoft\Windows\Start Menu\Programs\SoftwareKey Licensing System\Protection PLUS 5*), make a copy of the shortcut, and name the new shortcut so it contains the new profile name. (License Manager (Sample Name).lnk for the example shown here.)
3. Right-click on your new shortcut, click Properties, go click in the "Target" field, and add the resourcePath switch and the full path to the new folder you created to the end of it's current value. In this example, we will add the following to the end:
   /resourcepath "%PUBLIC%\Documents\SoftwareKey\Protection PLUS 5\License Manager\Sample Name"

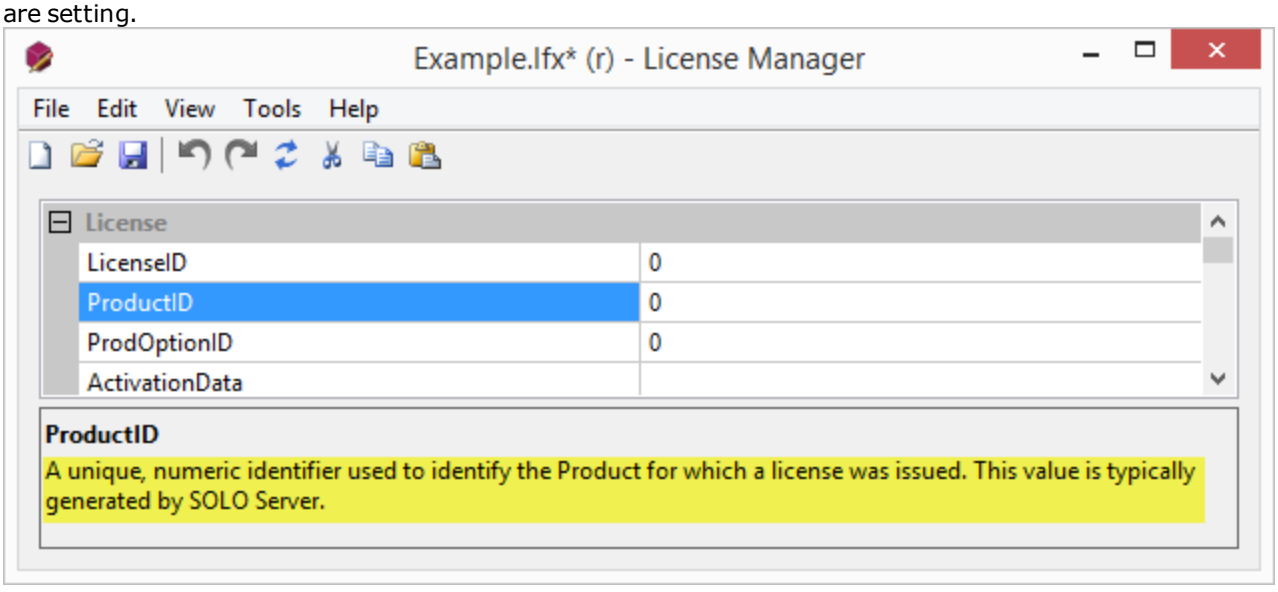
With this set, you can now go to your start screen or menu and click on your new shortcut. (You may need to type License Manager at the start screen or menu to find the new shortcut.) This will launch License Manager with your new profile, which you can configure using the *Tools / Options* menu to set your alternate encryption key data, profile name, etc.

To make sure you know which profile is currently being used, click the *Tools / Options* menu, and set the profile name. This name will appear in the title bar of License Manager. The profile name may only be configured when using the /resourcePath switch to use a different profile.



Changing the Field descriptions

Whenever you click on a license field in the License Manager interface, it lists a description at the bottom of the window. You can customize the description of each field to better describe your particular use of that field. This can be especially useful when training new employees to use License Manager and better understand the values that they are setting.



To change this description, edit the GridFieldLayout.xml file you copied into :*"%PUBLIC%\Documents\SoftwareKey\Protection PLUS 5\License Manager\Sample Name"* (see above for setting up a new profile). You can change the description for a particular field by editing the text inside the `<HelpString>` tag for that `<Field>`:

XML

```
<Field>
   <XPath>/SoftwareKey/PrivateData/License/ProductID</XPath>
   <Type>Int</Type>
   <Min>0</Min>
   <Required>True</Required>
   <HelpString>Change this to your own description.</HelpString>
</Field>
```

Important

You should only consider making any changes to GridFieldLayout.xml if you are comfortable with XML.

While you can add new, custom fields, you should never delete anything. Additionally, you may only modify the `HelpString` when editing existing/standard fields (altering anything else will break compatibility with Protection PLUS 5 SDK). Any customizations cannot be officially supported, and could have the potential to be incompatible with future versions of License Manager and Protection PLUS 5 SDK. As always, we recommend keeping back-ups of your GridFieldLayout.xml file(s) if you do make any additions (keeping track of changes using a Version Control System is an ideal way to do this).

Adding custom fields

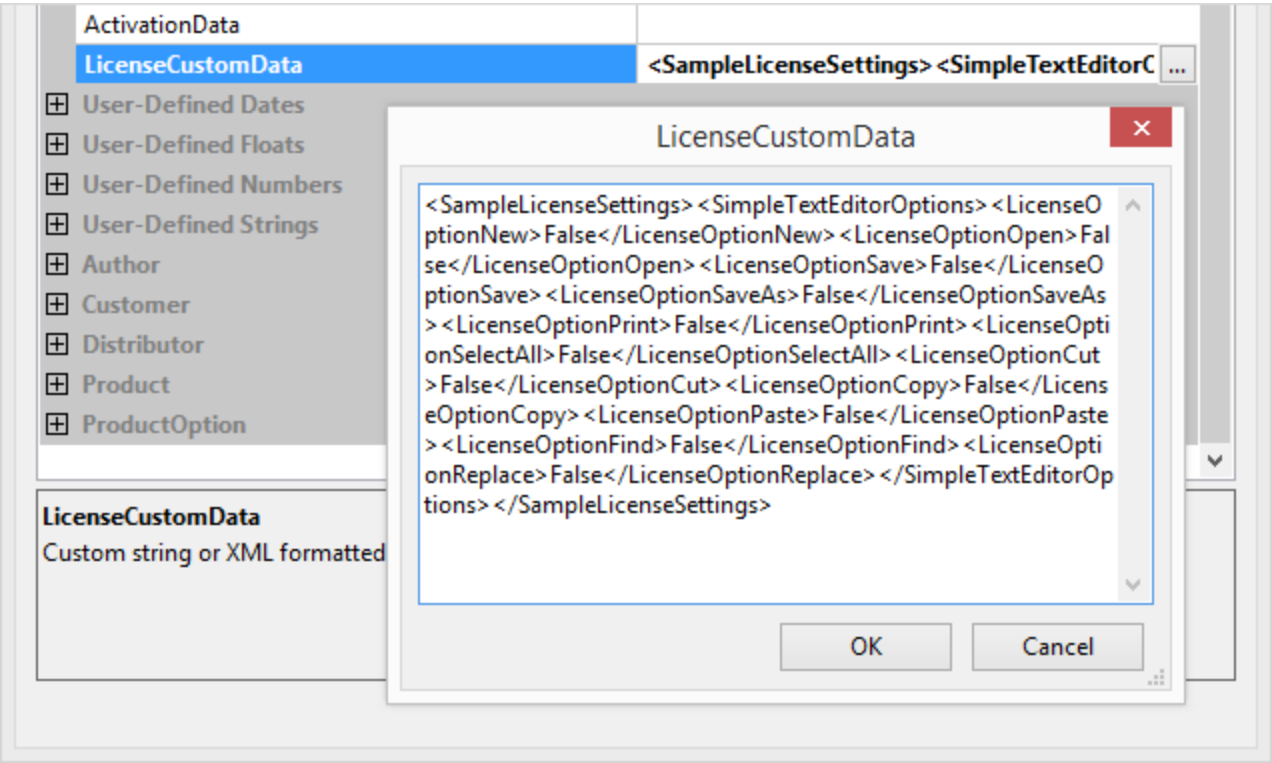
It is possible to add custom fields to License Manager, which will then be available in the license files. This brief guide will use the SimpleTextEditor sample from the PLUSManaged API to demonstrate how this functionality can be used for feature-based licensing.

The SimpleTextEditor sample has the following menu options that can be enabled or disabled: New, Open, Save, Save As, Print, Select All, Cut, Copy, Paste, Find, Replace. The license file needs to contain data that the application will read so it knows whether to enable or disable these features. The SimpleTextEditor sample is configured to enable/disable the menu option based on the `LicenseCustomData` field using a block of custom XML. The following would disable all menu options:

XML

```
<SampleLicenseSettings>
 <SimpleTextEditorOptions>
   <LicenseOptionNew>False</LicenseOptionNew>
   <LicenseOptionOpen>False</LicenseOptionOpen>
   <LicenseOptionSave>False</LicenseOptionSave>
   <LicenseOptionSaveAs>False</LicenseOptionSaveAs>
   <LicenseOptionPrint>False</LicenseOptionPrint>
   <LicenseOptionSelectAll>False</LicenseOptionSelectAll>
   <LicenseOptionCut>False</LicenseOptionCut>
   <LicenseOptionCopy>False</LicenseOptionCopy>
   <LicenseOptionPaste>False</LicenseOptionPaste>
   <LicenseOptionFind>False</LicenseOptionFind>
   <LicenseOptionReplace>False</LicenseOptionReplace>
 </SimpleTextEditorOptions>
</SampleLicenseSettings>
```

Updating or modifying the XML directly is not ideal, as the text box does not format the XML.



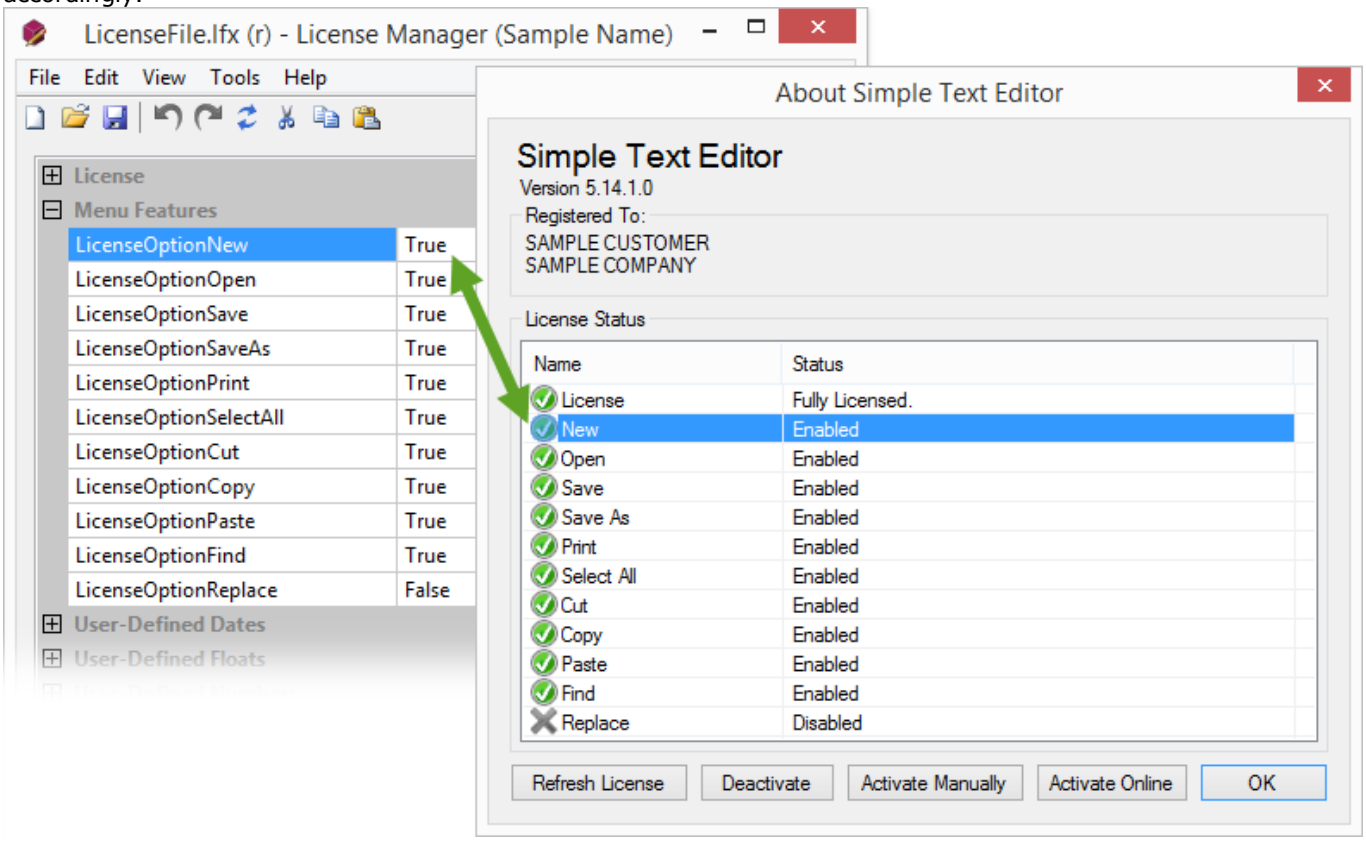
To address this, you can add custom fields for each feature into the License Manager interface. These fields will use the same XPaths as the data in the `LicenseCustomData` field, and will make it easier to set individual features as opposed to editing the XML directly.

To add the custom *Menu Features* category and `LicenseOption*` fields to License Manager, make the following additions to GridFieldLayout.xml after the *License* category and before the *User-Defined Dates* category:

XML

```
<!-- Custom Fields -->
<Field>
   <Name>Menu Features</Name>
   <Type>Category</Type>
   <HelpString>Menu features for the SimpleTextEditor PLUSManaged sample.</HelpString>
</Field>
<Field>
   <XPath>/Soft-
   wareKey/Priv-
   ateData/License/LicenseCustomData/SampleLicenseSettings/SimpleTextEditorOptions/LicenseOptionNew
   </XPath>
   <Category>Menu Features</Category>
   <Type>Bool</Type>
   <HelpString>Enable or disable the New menu option.</HelpString>
</Field>
<!-- Create similar entries for each menu option -->
```

Once you save the license file, the SimpleTextEditor sample will read this license file and enable/disable features accordingly.



PLUSManaged Overview

If your application is a managed, .NET application (written in C# or Visual Basic .NET, for example), then PLUSManaged is the API designed for your needs (see more about **selecting a solution**)

This section of the manual is designed to guide you or your application developers through the process of using the PLUSManaged API in your applications. Understanding this portion of the manual and using the API effectively requires at least a basic understanding of .NET application development and object-oriented programming (including inheritance and polymorphism), and also requires a basic understanding of the **SoftwareKey System™** and **licensing**. Topics in this section include subjects like **adding a reference to your project**, implementing **licensing requirements** in your application (such as **adding copy protection**, **validating time-limited licenses**, and much more), and **deploying your protected applications**.

Important

The steps laid out in the common implementation sub-topics of this manual do not match exactly what is in the **Protection PLUS 5 SDK sample projects**. The manual topics are more simplified to make it easier to explain, while the sample projects have more complex organization, but the resulting functionality is similar in both cases.

PLUSManaged: Features & Compatibility

The PLUSManaged library has an extensive set of features, and some of these features are specific to or limited to specific operating systems.

Feature	Windows	macOS	Linux
General Features			
Copy protection	✔	✔	✔
Read-only and writable **license files**	✔	✔	✔
Automatically sets privileges on license files	✔		
Activation & Electronic License Management (ELM)			
Online **activation**, **background checking**, and **deactivation**.	✔	✔	✔
Manual activation	✔	✔	✔
Manual activation via trigger codes	✔	✔	✔
SOLO Server Integration APIs	✔	✔	✔
System Identification Algorithms			
BIOS UUID	✔		
Computer name	✔	✔	✔
Domain name	✔		
*Hard disk volume serial	✔	✔	✔
Network Interface Card (NIC)	✔	✔	✔
Processor Information (name, vendor, version)	✔		

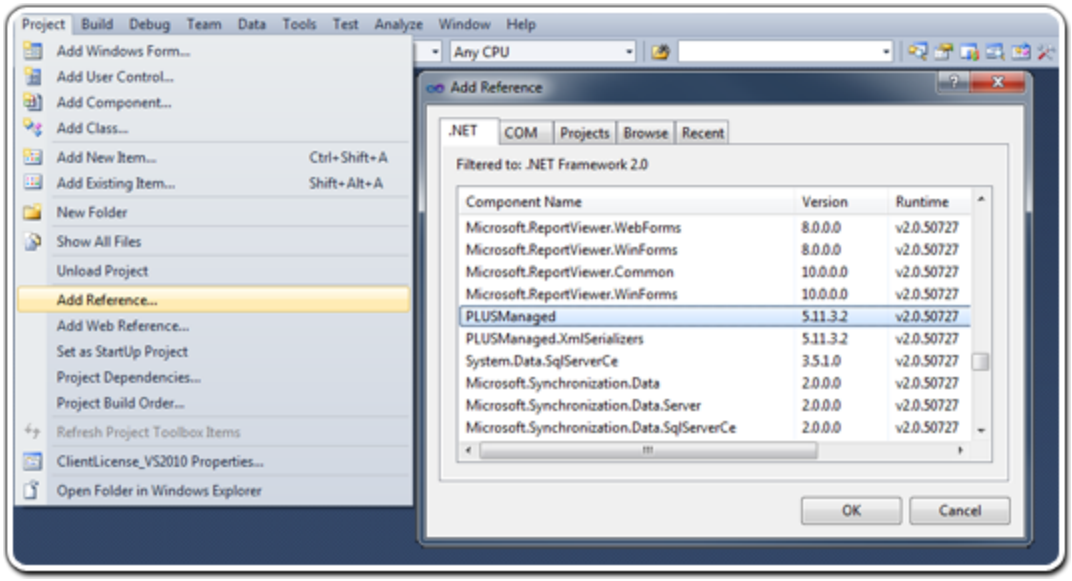
Server host name	✔		
User name	✔	✔	✔
Advanced Licensing Features			
Virtual machine guest environment detection	✔	✔	✔
Windows Terminal Services / Remote Desktop detection	✔		
Network floating licensing via semaphore files	✔		
Cloud-Controlled Floating Network Licensing using SOLO Server	✔	✔	✔
Volume licenses	✔	✔	✔
Downloadable licenses with trigger code activation	✔	✔	✔

*On macOS, the hard disk volume serial algorithm is only supported while targeting the 4.0 (v4.0.30319) framework or later, and requires the presence of the MonoMac.dll assembly to function. Any attempt to use it without this assembly present, or while targeting older framework versions, will result in unhandled exceptions being thrown.

Adding PLUSManaged to your application

Before you can use PLUSManaged with your own application, you need to add a reference to the assembly in your application's project.

In Visual Studio, begin by clicking *Project/Add Reference* from the menu. Then, click the .NET tab, select PLUSManaged from the list, and click OK.



PLUSManaged: Creating the License Implementation

PLUSManaged is an object oriented API, and is designed to make heavy use of features specific to object-oriented programming. The most important feature of object oriented programming used is inheritance, as PLUSManaged provides you with an abstract class from which your implementation will be derived. Of course, the first steps to using PLUSManaged in your assembly includes **defining your licensing requirements** and **adding a reference** to the PLUSManaged assembly.

Next, you will need to decide what to name your licensing implementation class. The name of your class can reflect the product being licensed (i.e. `XyzProductLicense`); the type of license file used (i.e. `ReadOnlyLicense`); or, in some more advanced implementations, the type of license (i.e. `EvaluationLicense` or `NetworkFloatingLicense`). It is ultimately up to you to make the best choice based on what is most appropriate for your application. For the purposes of this guide, we will name the implementation class `SampleLicense`.

Important

The steps laid out in the common implementation sub-topics of this manual do not match exactly what is in the **Protection PLUS 5 SDK sample projects**. The manual topics are more simplified to make it easier to explain, while the sample projects have more complex organization, but the resulting functionality is similar in both cases.

Namespaces

Like most object oriented APIs, PLUSManaged groups its objects into categorically organized namespaces. Your license implementation class will need to specify that it uses/imports several namespaces as outlined in the code example below:

C#

```
using System;
using System.Collections.Generic;
using System.IO;
using com.softwarekey.Client.Licensing;
using com.softwarekey.Client.Utils;
using com.softwarekey.Client.WebService.XmlActivationService;
using com.softwarekey.Client.WebService.XmlLicenseFileService;
```

VB.NET

```
Imports System
Imports System.Collections.Generic
Imports System.IO
Imports com.softwarekey.Client.Licensing
Imports com.softwarekey.Client.Utils
Imports com.softwarekey.Client.WebService.XmlActivationService
Imports com.softwarekey.Client.WebService.XmlLicenseFileService
```

The System, System.Collections.Generic, and System.IO namespaces are part of the .NET Framework, and may already be in your source code. If they are, you will need to make sure there is only a single using/Imports statement at the top of your class file for each. The com.softwarekey.Client namespaces are the ones which contain objects in PLUSManaged. Although the namespaces are **documented in the PLUSManaged API reference**, here is a brief summary of the namespaces used here:

Namespace	Description
com.softwarekey.Client.Licensing	Contains objects and data for **licensing**, including the base License (for read-only licenses) and WritableLicense (for writable, self-signed license files).
com.softwarekey.Client.Utils	Contains utility objects which are designed to help with a variety of common tasks. This includes things such as XML parsing and manipulation, file system functions, encryption data management, and more.
com.softwarekey.Client.WebService.XmlActivationService	Contains objects required to submit and process activation and other **ELM** functions.
com.softwarekey.Client.WebService.XmlLicenseFileService	Contains objects required to perform background checking and license file refreshing simultaneously.

Adding the namespaces to your class file using the using/Imports statements allows you to reference the classes directly without the need to reference it using the full namespace. So for example, without the using/Imports statements, you would reference the com.softwarekey.Client.Utils.IOHelper class as shown here; however, you can simply reference it directly as IOHelper with these statements present. This helps make your application's code a little easier to read and maintain. Also keep in mind that it is possible that your application or an assembly it references and uses could use a class name which matches that of one of the classes used from PLUSManaged. When you encounter an ambiguous reference like this, your application will likely generate compiler errors or warnings, which can easily be resolved by referencing the class using the full name space (i.e. com.softwarekey.Client.Utils.IOHelper).

Inheriting from PLUSManaged

Next, the PLUSManaged API provides you with two abstract classes from which your license class should be derived. The class you will use will depend on the type of **license files** you intend to use. If you intend to use **writable, self-signed license files**, your class will need to inherit from WritableLicense. Otherwise, if you intend to use the more secure, read-only license files issued by SOLO Server, your class must inherit from the License class. Our example will use the latter, as shown in the example code excerpt below.

C#

```
public class SampleLicense : License
{
}
```

VB.NET

```
Public Class SampleLicense
   Inherits License
End Class
```

Implementing Licensing Requirements

Now that the initial configuration of your class is established, you can begin implementing your licensing class. Before doing this, it is important to have already defined your **licensing requirements**, as well as how your application needs to **identify a system**. For the example used in this guide, we will use network interface cards (NICs), hard drive volume format serials, and the computer name to identify the system being licensed.

Adding a Constructor

The first thing you must add to make your class usable is a constructor. The constructor is what gets called to create a new instance of your license implementation class. Below is an example excerpt of a constructor for the `SampleLicense` class, which passes required data to the base `License` constructor.

C#

```
public SampleLicense()
   : base(new AuthorEncryptionKey("[Encryption Envelope Key", "[Encryption Envelope]"),
     true, true, [Product ID], "[Product Version]",
     new List<SystemIdentifierAlgorithm> (new SystemIdentifierAlgorithm[] {new NicIden-
     tifierAlgorithm(), new ComputerNameIdentifierAlgorithm(), new HardDiskVolumeSeri-
     alIdentifierAlgorithm()}))
{
}
```

VB.NET

```
Public Sub New()
   MyBase.New(New AuthorEncryptionKey("[Encryption Envelope Key]", "[Encryption Envelope]"),
   _
     True, True, [Product ID], "[Product Version]", _
     New List(Of SystemIdentifierAlgorithm)(New SystemIdentifierAlgorithm() {New NicIden-
     tifierAlgorithm(), New ComputerNameIdentifierAlgorithm(), New HardDiskVolumeSeri-
     alIdentifierAlgorithm()}))
End Sub
```

Important

The code excerpts above are just examples, which must have several pieces of information populated!

Here is how to populate the required data in the example code excerpt above.

- **Your encryption key data** (includes `[Encryption Envelope Key]` and `[Encryption Envelope]`) must be passed in using an `AuthorEncryptionKey` object.
- The `[Product ID]` must be replaced with the value appropriate **Product ID from SOLO Server**.
- The `[Product Version]` must be removed or replaced with the version of your application. If used, the format must be like *x.x.x.x* (where each *x* is a 1 to 5 digit, positive integer).
- The `SystemIdentifierAlgorithm` implementations listed here are only present to show you how you can initialize the list of algorithms your application uses. If your application needs to use a different combination of identifiers, you will need to change this list.

In the **PLUSManaged samples** and the **PLUSManagedGui samples**, these values are all defined in a `LicenseConfiguration` class for convenience. You can copy and modify this class as needed as an alternative to defining the values directly in your constructor.

Add a License Validation Method

The next step is to get a license validation method in place, which will be called by your application's code any time it needs to validate its license. This typically needs to be called any time your application is launched by a user, and when it performs some kind of important function. For example, if your application takes some input from a user and uses that to generate some kind of document as the output, then it is good practice to validate the license before generating the output. Here is a code excerpt that only gets the method in place (it is **not** a complete implementation!):

C#

```
public bool Validate()
{
   //TODO: Add copy protection logic!
   return true;
}
```

VB.NET

```
Public Function Validate() As Boolean
   'TODO: Add copy protection logic!
   Return True
End Function
```

Before **adding copy protection** logic to this method, you need to add some code to **activate and get a license file**.

PLUSManaged: Calling SOLO Server XML Web Services

Important

If you are using or planning to use **PLUSManagedGui** in your application, this content may not be relevant to you since PLUSManagedGui automatically handles common functionality.

There are a variety of ways to call SOLO Server web services in your .NET application. The best method of calling to a web service depends on whether or not you are calling it from your applications license implementation class, and whether or not helper methods or classes to call the relevant web service method are available.

Making Calls From Your License Implementation

The most common way to call to SOLO Server web services with PLUSManaged is using helper methods which are automatically inherited (from `License` or `WritableLicense`) in your **license implementation class**. The methods listed below are available to your license implementation class, and are used throughout the **sample applications**, which may be used as a supplement to see how these methods are called in-practice.

Method	Description
ActivateInstallationLicenseFile	Activates a license online and retrieves the license file.
ActivateOnline	Activates a license online. This does not retrieve the license file, but it does retrieve the Installation ID that can be used with **RefreshLicense** to do so.
CheckInstallationStatus	Performs a light-weight background check of the activated license.
DeactivateInstallation	Deactivates the license which was previously activated.
RefreshLicense	Performs a background check of the activated license and retrieves the most current license file from SOLO Server.

Using the WebServiceCall Implementations

You might have an application which already has a reference to PLUSManaged, but needs to call to a SOLO Server outside of the license implementation class. If your application has this sort of requirement, PLUSManaged fulfills it via the WebServiceCall implementations. Refer to the **PLUSManaged API reference** for additional details and example source code. The inheritance hierarchy at the bottom of the API reference shows a list of all `WebServiceCall` implementations available to you.

Calling Web Services Directly

SOLO Server offers a wide variety of web services, and PLUSManaged may not have helper methods or classes for all of the methods you might want to use. However, you can easily add **a web service reference** to your application, and reference the **SOLO Server manual** for details on how to call each web service method directly (including the web service's endpoint URL, which you will need when adding the reference).

PLUSManaged: Activating and Getting a License File (Online Method)

Online activation is one option for activating an application and retrieving a license file from SOLO Server. Obtaining a license file from SOLO Server is required when working with read-only license files, since only SOLO Server has the encryption key data necessary to encrypt and digitally sign a read-only license file. Although it is possible to create a self-signed writable license file without using SOLO Server, it is often more convenient to obtain the base license file from SOLO Server that can then be modified as necessary. Either way, the procedure is largely the same. The only difference being the class from which your license implementation class is derived.

Important

Activation is only intended to occur once per system and/or user. DO NOT configure your application to activate automatically every time it is run, as this can cause many problems that affect the reliability of your application, and is considered a misuse of the SOLO Server activation web services. If you wish to validate the status of the license with SOLO Server in your application, use background checking. If you wish to offer a limited number of users an option to check-in/check-out seats for a license, consider using **Network Floating Licensing** features for this purpose.

The following guide extends on the **example license implementation** to demonstrate activating an installation, retrieving a license file from SOLO Server, and then saving that license file to disk. For all example code excerpts, `[License File Path]` should be replaced with the actual path to your license file (which you may store in a constant or property in your license implementation class).

Online Activation

Important

If you are using or planning to use **PLUSManagedGui** in your application, this content may not be relevant to you since PLUSManagedGui automatically handles common functionality.

PLUSManaged has different ways of calling to the SOLO Server web services. Typically, you will only need to perform activation from your license implementation code, which inherits from `License` or `WritableLicense`. `License` and `WritableLicense` both contain methods which simplify calling SOLO Server web services. The example code excerpt below illustrates how your application's license implementation class may leverage these convenience methods to support activation.

C#

```
public bool ActivateOnline(Int32 licenseId, string password)
{
   //initialize the object used for calling the web service method
   XmlActivationService ws = new XmlActivationService();

   string lfContent = "";
   if (!this.ActivateInstallationLicenseFile(licenseId, password, ws, ref lfContent))
      return false;


   //try to save the license file to the file system
   try
   {
      File.WriteAllText("[License File Path]", lfContent);
   }
   catch (Exception ex)
```

```
   {
      this.LastError = new LicenseError(LicenseError.ERROR_COULD_NOT_SAVE_LICENSE, ex);
      return false;
   }

   return true;
}
```

VB.NET

```
Public Overloads Function ActivateOnline(ByVal licenseId As Int32, ByVal password As String)
As Boolean
   'initialize the object used for calling the web service method
   Dim ws As New XmlActivationService()

   Dim lfContent As String = ""
   If Not Me.ActivateInstallationLicenseFile(licenseId, password, ws, lfContent) Then
      Return False
   End If

   'try to save the license file to the file system
   Try
      File.WriteAllText("[License File Path]", lfContent)
   Catch ex As Exception
      Me.LastError = New LicenseError(LicenseError.ERROR_COULD_NOT_SAVE_LICENSE, ex)
      Return False
   End Try

   Return True
End Function
```

PLUSManaged: Manual Activation (Offline Method)

Important

If you are using or planning to use **PLUSManagedGui** in your application, this content may not be relevant to you since PLUSManagedGui automatically handles common functionality.

When Internet access is not available and online activation is not possible, manual activation is the recommend method for activating an installation and obtaining a license file from SOLO Server. This involves the end-user copying the activation request or saving it to a file that they would take to another computer with Internet access. They would then paste the request or upload the file to SOLO Server's manual request page, which processes the request and generates a response. The end-user must then copy this response or save it to a file and return to the computer they are attempting to activate. Finally, they must paste in the response or point the application to the file. The application can then parse the response and determine which action to take.

The following demonstrates manually activating an installation, retrieving a license file from SOLO Server, and saving that license file to disk.

Generating and Encrypting the Request

The PLUSManaged library has built-in functions for generating requests for the SOLO Server **XML Activation Service** and **XML License File Service** web services. Use of these functions avoids having to manually build the web service XML request document. If necessary, further control over generation of the requests may be achieved via the `WebServiceCall` **classes**.

The following code snippet demonstrates generating the manual activation request:

C#

```
string request = m_License.GetActivationInstallationLicenseFileRequest(licenseID, password);
```

VB.NET

```
Dim request As String = m_License.GetActivationInstallationLicenseFileRequest(licenseID, password)
```

This excerpt of code assumes `m_License` is a reference to an instance of your applications `License` or `WritableLicense` implementation class, `licenseID` is an `Int32` object obtained from the user, and `password` is a `string` obtained from the user. Now your application may allow the user to copy the request (so it may be pasted to SOLO Server's manual request page), or it may allow the user to save the file (which may be uploaded via SOLO Server's manual request page).

Using the SOLO Server Manual Request Page

Once the user generates the manual activation request, they must take it to a computer with Internet access where they can paste it in or upload it to the **SOLO Server manual request page**. If SOLO Server is able to successfully process the manual request it will generate a response that must be taken back to the computer being activated. Otherwise, it will display an error message and it may be necessary for the end-user to contact support for help and possibly have to start the process over.

Important

The above SOLO Server Manual Request link is for use with Instant SOLO Server only. If you are using a dedicated or externally-hosted SOLO Server, this will need to be updated to point to your specific domain.

Manual Request HTML Files

Manual request HTML files are convenient for your users since the files can help automate submitting manual requests. In short, these files can be generated easily with PLUSManaged using the `ManualRequestFile` **class**. The file will contain the generated request, and can automatically post the request to SOLO Server's manual request page. The code example below shows how you can use this class to automatically save a file to the user's desktop, which he or she may then copy to another computer, and then double click it to process the request using the default web browser.

C#

```
string requestFilePath = Path.Combine(Environment.GetFolderPath
(Environment.SpecialFolder.Desktop), "ActivateXyzProduct.html");
ManualRequestFile requestFile = new ManualRequestFile(request, requestFilePath);
requestFile.Save();
```

VB.NET

```
Dim requestFilePath As String = Path.Combine(Environment.GetFolderPath(Environment.Spe-
cialFolder.Desktop), "ActivateXyzProduct.html")
Dim requestFile As New ManualRequestFile(request, requestFilePath)
requestFile.Save()
```

The code provided above should be modified to specify a file name other than `"ActivateXyzProduct.html"` (so that it reflects the name of your application instead), and it can also be updated to use a path selected by the user as well (thus allowing the user to use a browse dialog to specify the location instead, if preferred).

Important

If the user is pasting in a response instead of loading a response file, there are a few potential issues to be aware of:

- Users may be limited by the size of their clipboard if the response is too large. An"Invalid Data" error will occur if the entire response is not pasted in.
- For macOS users, if the encrypted response contains the characters "TM" and the user pastes in the response as a string, the operating system will automatically convert the "TM" characters to the trademark superscript character, which invalidates the response.

These issues do not occur when the encryption response is loaded from a file.

Decrypting and Parsing the Response

After the end-user returns to the computer being activated with the encrypted response, it must be decrypted and verified.

C#

```
string lfContent = "";
bool successful = m_License.ProcessActivateInstallationLicenseFileResponse(response,
ref lfContent);
```

VB.NET

```
Dim lfContent As String = ""
Dim successful As Boolean = m_License.ProcessActivateInstallationLicenseFileResponse
(response, lfContent)
```

In the above code, it is assumed that `m_License` is an instance of your `License` or `WritableLicense` implementation class, and `response` contains the response received from SOLO Server's manual request page. Once you create an XML document from the license string you can save it to disk. If the response is successfully decrypted and verified, `successful` will be set to `true`, and you may then save the value of `lfContent` to disk as your application's new license file. Otherwise, if the response could not be decrypted or verified or indicates another error, `successful` will be set to `false`, and additional details about the error will be available in the `m_License.LastError` property.

Important

These basic source code examples omit important and necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it is important that you make sure each function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where a problem originated if and when one occurs.

Session Code Validation

Session code validation to prevents the end-user from processing the same response multiple times, which is known as a replay attack. A random session code is generated and included in each web service request. This session code must be saved somewhere on the system so it can later be retrieved and compared to the session code found in the web service response. If the session codes match then you know the response was the response associated with the original web service request. Otherwise, the response should be consider invalid. Your license implementation class will inherit the `CurrentSessionCode` property and the `ResetSessionCode` method from `License` or `WritableLicense`. You may use the `CurrentSessionCode` property to read the session code value (to store a session code after generating a request), and you can set the property to restore a prior value before processing a response. The `ResetSessionCode` method generates a new session code, and should always be called after successfully processing a response. Additionally, you may opt to be a little more strict and do this when any response processing is attempted, as doing so would be more secure (although it means customers have to generate a new request after each attempt, even if it fails, which could pose some inconvenience depending on how much effort is involved with the customer using a different computer to activate).

Using Trigger Codes for Telephone Activations

Trigger code validation is a means of achieving software activation through the manual exchange of numeric values (in other words, it is a challenge-response mechanism). Review the **overview on trigger codes** for additional, high-level details on how this works and how to generate activation codes.

Important

Keep in mind that any challenge-response mechanisms such as trigger code validation can be reverse engineered (similar to how "key generators" are available on the Internet for many popular software applications). Therefore, it is best to use Protection PLUS 5 SDK's standard **online** and **manual** activation features when possible, which offer much stronger **security** than trigger codes.

Adding Support for Trigger Codes to your Application

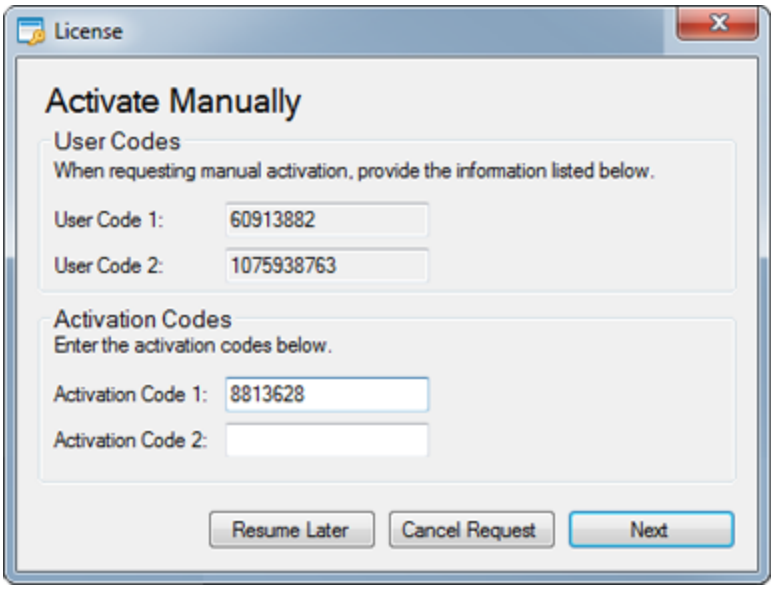
Before you can add support for any form of activation, your application must first have a **license implementation**. Once activation has been implemented, the application will also need to include **copy protection** to ensure it only runs on devices in which it was activated.

Creating an Activation Form

Important

If you are using or planning to use **PLUSManagedGui** in your application, this content may not be relevant to you since PLUSManagedGui automatically handles common functionality.

To process trigger code activations, you will need to create a form or dialog that will make this kind of activation available to your users. Most activation forms are pretty similar, and can be designed similar to the illustration below.



As a convenience, you can use/import the namespace where the Protection PLUS 4 compatibility methods are located at the top of your relevant source file(s):

C#

```
using com.softwarekey.Client.Compatibility.ProtectionPLUS4;
```

VB.NET

```
Imports com.softwarekey.Client.Compatibility.ProtectionPLUS4
```

Next, you would implement your activation form, which would have variables, labels, and text boxes for the following values: User Code 1, User Code 2, Activation Code 1, and Activation Code 2. As part of the example code snippets included in this walk-through, we will make the following assumptions:

- The form has member variables (of type `Int32`) for the User Code 1, User Code 2, Activation Code 1, and Activation Code 2 values. These would be named `m_UserCode1`, `m_UserCode2`, `m_ActivationCode1`, and `m_ActivationCode2`, respectively.
- The form also has some constants defined for the required Trigger Code Seed and RegKey2Seed values, named `m_TriggerCodeSeed` and `m_RegKey2Seed`, respectively. Note that it is important to make a reasonable effort to hide/obfuscate these values to prevent hackers from finding them. Most obfuscation tools do encrypt constants, but many only encrypt strings (and you could store them as strings and convert them for this purpose).
- The form's `TextBox` controls for the User Code 1, User Code 2, Activation Code 1, and Activation Code 2 fields are named: `userCode1TextBox`, `userCode2TextBox`, `activationCode1TextBox`, and `activationCode2TextBox`, respectively.
- An `m_License` member variable is also defined in your form, and is your **license implementation class** object.

Of course, you can change the names actually used on your forms at your own discretion. However, we do strongly recommend making sure the text boxes for the User Code 1 and User Code 2 fields are read-only. Additionally, when processing the trigger codes, you want to make sure you use your own member variables for the user code values instead of parsing these values from the text boxes (for security purposes).

Since the User Code values will be relayed to you by your end-user, your new activation form will need to initialize the values displayed on the form. (the Activation Code fields should initially be blank). You can generate the User Code values as shown in the example below, which assumes `m_License` is your **license implementation class** object, and also assumes that some `TextBox` controls have been created on the form.

C#

```
m_UserCode1 = PLUS4Methods.GenerateUserCode1Value();
m_UserCode2 = PLUS4Methods.GenerateUserCode2Value(m_License.CurrentIdentifiers);
userCode1TextBox.Text = m_UserCode1.ToString();
userCode2TextBox.Text = m_UserCode2.ToString();
```

VB.NET

```
m_UserCode1 = PLUS4Methods.GenerateUserCode1Value()
m_UserCode2 = PLUS4Methods.GenerateUserCode2Value(m_License.CurrentIdentifiers)
userCode1TextBox.Text = m_UserCode1.ToString()
userCode2TextBox.Text = m_UserCode2.ToString()
```

Processing the Activation

Then, when the user clicks the a button (which usually says "OK" or "Activate"), your form would read the values from the fields, and convert the Activation Code 1 and Activation Code 2 values entered by the user to 32 bit integers (you can use `Int32.TryParse` or `Int32.Parse` to do this). Once your form has successfully validated and converted these values, it could then call a function similar to the one below to process the trigger code activation.

C#

```
internal void ProcessTriggerCode()
{
   Int32 triggerCodeNumber = 0;
   Int32 eventData = -1;

   if (false == PLUS4Methods.ValidateTriggerCode(m_ActivationCode1, m_ActivationCode2, m_User-
   Code1, m_UserCode2, m_TriggerCodeSeed, m_RegKey2Seed, out triggerCodeNumber, out
   eventData))
   {
      //TODO: The activation codes failed validation. Add error handling here.
   }
   else
   {
      //The activation codes were validated successfully.
      //TODO: Add your trigger code processing here! A rough example is outlined below.
      switch (triggerCodeNumber)
      {
         case 1:
            //As an example, code could be added here so that trigger code 1 activates your
            application as a full, non-expiring (or perpetual) license.
            break;
         case 10:
            //As an example, code could be added here so that trigger code 10 activates your
            application as a time-limited (or lease/periodic) license.
            //You could use the value in eventData to determine the number of days until the
            license expires by setting your license's EffectiveEndDate property to
            DateTime.UtcNow.AddDays(eventData).
            break;
         default:
            //TODO: An unsupported trigger code number was specified. Add error handling here.
            break;
      }
   }
}
```

VB.NET

```
Friend Sub ProcessTriggerCode()
   Dim triggerCodeNumber As Int32 = 0
   Dim eventData As Int32 = -1

   If False = PLUS4Methods.ValidateTriggerCode(m_ActivationCode1, m_ActivationCode2, m_User-
   Code1, m_UserCode2, m_TriggerCodeSeed, m_RegKey2Seed, triggerCodeNumber, eventData) Then
      'TODO: The activation codes failed validation. Add error handling here.
   Else
      'The activation codes were validated successfully.
      'TODO: Add your trigger code processing here! A rough example is outlined below.
      Select Case triggerCodeNumber
         Case 1
            'As an example, code could be added here so that trigger code 1 activates your
            application as a full, non-expiring (or perpetual) license.
         Case 10
```

```
            'As an example, code could be added here so that trigger code 10 activates your
            application as a time-limited (or lease/periodic) license.
            'You could use the value in eventData to determine the number of days until the
            license expires by setting your license's EffectiveEndDate property to
            DateTime.UtcNow.AddDays(eventData).
         Case Else
            'TODO: An unsupported trigger code number was specified. Add error handling here.
      End Select
   End If
End Sub
```

While the code snippet above only gives high-level guidance on how to implement trigger code processing, the PLUSManagedGui samples include fully-functioning source code that supports trigger code processing for various types of licenses (including perpetual and time-limited licenses).

Delayed Trigger Codes

Important

If you are using or planning to use **PLUSManagedGui** in your application, this content may not be relevant to you since PLUSManagedGui automatically handles common functionality.

A *delayed trigger* code is simply where you allow the user to process a trigger code activation after having closed and re-launched your application. The problem this solves primarily revolves around how the activation form initializes the User Code 1 value, as this value is randomized. Although this is not typically an issue with telephone activations, this can pose a challenge when the user might email, fax, or text the user code values, close the application, and re-launch the application to complete the activation later. If your application needs to support this kind of delay between when the user sends the user code values and when the user completes the activation, your application may need to implement this feature by storing the user code 1 value somewhere. This can be stored in a hidden file or registry key of your choosing, and it is strongly recommended that you consider encrypting the value as well.

Storing the User Code 1 Value

One way to store the User Code 1 value is to simply store it in your license file. The only drawback to this approach, however, is that it may be possible that issues could arise with this approach when allowing users to refresh online (when an Installation ID is present in the license file), and when using aliases (which could restore old values to the license file). Consequently, storing this value separately from the license file is the preferred approach.

Below is an example of how you could store the user code 1 value in the registry. (Note that you should consider using encryption as well if you decide to use this approach.)

C#

```
private void InitializeUserCode1()
{
   string keyPath = @"Software\Company Name\Application Name";
   string keyValue = "SessionCode";
   string sessionCode = "";

   //Open the registry key for reading so we may try to retrieve a previously saved value.
   using (RegistryKey key = Registry.CurrentUser.OpenSubKey(keyPath, false))
   {
      if (null != key)
      {
```

```
            sessionCode = (string)key.GetValue(keyValue);
            int.TryParse(sessionCode, out m_UserCode1);
            key.Close();
         }
      }
      if (0 == m_UserCode1)
      {
         m_UserCode1 = PLUS4Methods.GenerateUserCode1Value();

         //Open the registry key for writing so we may store the value for later use.
         using (RegistryKey wkey = Registry.CurrentUser.OpenSubKey(keyPath, true))
         {
            if (wkey != null)
            {
               wkey.SetValue(keyValue, m_UserCode1.ToString(), RegistryValueKind.String);
               wkey.Close();
            }
            else
            {
               //The key does not exist, so try to create it.
               using (RegistryKey newKey = Registry.CurrentUser.CreateSubKey(keyPath))
               {
                  if (newKey != null)
                  {
                     newKey.SetValue(keyValue, m_UserCode1.ToString(), RegistryValueKind.String);
                     newKey.Close();
                  }
               }
            }
         }
      }
   }
```

VB.NET

```
Private Sub InitializeUserCode1()
   Dim keyPath As String = "Software\Company Name\Application Name"
   Dim keyValue As String = "SessionCode"
   Dim sessionCode As String = ""

   'Open the registry key for reading so we may try to retrieve a previously saved value.
   Using key As RegistryKey = Registry.CurrentUser.OpenSubKey(keyPath, False)
      If Not key Is Nothing Then
         sessionCode = key.GetValue(keyValue)
         Integer.TryParse(sessionCode, m_UserCode1)
         key.Close()
      End If
   End Using

   If 0 = m_UserCode1 Then
      m_UserCode1 = PLUS4Methods.GenerateUserCode1Value()

      'Open the registry key for writing so we may store the value for later use.
      Using wkey As RegistryKey = Registry.CurrentUser.OpenSubKey(keyPath, True)
```

```
        If Not wkey Is Nothing Then
           wkey.SetValue(keyValue, m_UserCode1.ToString(), RegistryValueKind.String)
           wkey.Close()
        Else
           'The key does not exist, so try to create it.
           Using newKey As RegistryKey = Registry.CurrentUser.CreateSubKey(keyPath)
              If Not newKey Is Nothing Then
                 newKey.SetValue(keyValue, m_UserCode1.ToString(), RegistryValueKind.String)
                 newKey.Close()
              End If
           End Using
        End If
     End Using
  End If
End Sub
```

You may then replace the line of code that previously initialized `m_UserCode1` to call the method above. Of course, the above code is just an example, and it is strongly recommend that you specify your own registry location (unique to each application), and that you extend this example code to encrypt the value stored in the registry.

Clearing the User Code 1 Value

When storing the user code 1 value for alter use, it is pivotal that the stored value is cleared any time the user attempts to process activation codes, even if that attempt fails. This not only prevents users from re-using activation codes that were previously issued (which would be dangerous when an activation does something like increment the number of network seats, or days left for a license), but it also helps deter "brute-force" attacks (where a hacker or malicious script tries all possible values until it eventually gets the correct value).

Below is an example method that could be used to clear the value stored in the registry.

C#

```
private void ClearSessionCode()
{
   using (RegistryKey key = Registry.CurrentUser.OpenSubKey(@"Software\Company Name\Ap-
   plication Name"))
   {
      if (null != key)
      {
         key.DeleteValue("SessionCode");
          key.Close();
      }
   }
}
```

VB.NET

```
Private Sub ClearSessionCode()
   Using key As RegistryKey = Registry.CurrentUser.OpenSubKey("Software\Company Name\Ap-
   plication Name")
      If Not key Is Nothing Then
         key.DeleteValue("SessionCode")
         key.Close()
      End If
   End Using
End Sub
```

Your application would then call the above method after calling `PLUS4Methods.ValidateTriggerCode`, regardless of its result.

PLUSManaged: Background Checking and Refreshing Licenses

Checking the status of an activated license by **calling SOLO Server web services** is an effective and efficient way to check if activated licenses remain authorized. These checks can prevent copies of your application from running in scenarios like the following.

- A user purchases a license for your application and activates it, but later requests refund. When processing the refund, the license status should be changed in SOLO Server to indicate the protected application is no longer authorized to run.
- Similar to the above, a user purchases your application, but does so with fraudulent payment information, which could result in a charge-back. If the charge-back cannot be resolved in your favor, the license status should be changed in SOLO Server to indicate the protected application is no longer authorized to run.
- A user installs and activates on one computer, some time passes, and that computer experiences some kind of failure that renders the prior activation unusable according to the user. You may then log into SOLO Server, deactivate that computer's "Installation ID" so that specific computer is no longer authorized, and the user may then be allowed to activate on a new computer.

In each case, the background check will verify the system on which the application was activated is still authorized by verifying the status of the license and the individual system's activation are still valid. If the background check indicates the license is no longer authorized for the activated system, then your application can take steps needed to disable itself as needed. This is especially important if you allow users to **deactivate**.

Important

Be mindful about whether or not disabling access to your application is appropriate. For example, if your application requires high availability (such as a web application or service) and/or your application serves some mission-critical function, it may be more appropriate to have your application display nag messages and/or disable only non-critical features so as to avoid disrupting your customers' business or operation.

Important

When using background checking or license refreshing features, it is very important that you test this functionality in your application while running in an environment where connections are refused or unavailable, and also where connections time out. Doing so can help you identify areas of your application that may become unresponsive or unstable when your application encounters these conditions.

Background Checking Licenses

The XML Activation Service web service has a `CheckInstallationStatus` method, which your application can leverage to check the status of the license and the status of the system's activation. This is particularly ideal when your application needs to check its status frequently (such as every time the application runs, or every time a critical feature of your application is used if it is always running).

C#

```
public override bool CheckInstallationStatus()
{
   //initialize the object used for calling the web service method
   using (XmlActivationService ws = new XmlActivationService())
   {
      if (!base.CheckInstallationStatus(ws))
      {
         if (LastError.ExtendedErrorNumber == 5010 ||
```

```
            LastError.ExtendedErrorNumber == 5015 ||
            LastError.ExtendedErrorNumber == 5016 ||
            LastError.ExtendedErrorNumber == 5017)
         {
            File.Delete("[License File Path]");
         }
      }
      return false;
   }

   return true;
}
```

VB.NET

```
Public Overrides Function CheckInstallationStatus() As Boolean
   'initialize the object used for calling the web service method
   Using ws As New XmlActivationService()
      If Not MyBase.CheckInstallationStatus(ws) Then
         If LastError.ExtendedErrorNumber = 5010 Or
            LastError.ExtendedErrorNumber = 5015 Or
            LastError.ExtendedErrorNumber = 5016 Or
            LastError.ExtendedErrorNumber = 5017 Then
            File.Delete("[License File Path]")
         End If
         Return False
      End If
   End Using

   Return True
End Function
```

Note how this example code deletes the license when SOLO Server returns certain **result codes**. This is because the license should be considered invalid when the application receives one of these result codes from SOLO Server. Deleting the license file is an appropriate response to these result codes for read-only `License` implementations. However, when using `WritableLicense` implementations, you should alter the local license file in-code as appropriate for your licensing requirements, and save it and its aliases.

Refreshing Licenses

You may also implement license refreshing in your application. This allows your application to query SOLO Server for a license file with the latest licensing data, while also performing a background status check. This is ideal for periodically validating licenses (which helps deal with issuing refunds and with fraud when using e-commerce), and for updating license data periodically and/or on demand (which is very convenient for you and your customers with updating expiration on time-limited licenses).

C#

```
public bool RefreshLicense()
{
   string lfContent = "";
   //initialize the object used for calling the web service method
   using (XmlLicenseFileService ws = new XmlLicenseFileService())
   {
```

```
      if (!base.RefreshLicense(ws, ref lfContent))
      {
         if (LastError.ExtendedErrorNumber == 5010 ||
            LastError.ExtendedErrorNumber == 5015 ||
            LastError.ExtendedErrorNumber == 5016 ||
            LastError.ExtendedErrorNumber == 5017)
         {
            File.Delete("[License File Path]");
         }
         return false;
      }
   }

   //try to save the license file to the file system
   try
   {
      File.WriteAllText("[License File Path]", lfContent);
   }
   catch (Exception ex)
   {
      this.LastError = new LicenseError(LicenseError.ERROR_COULD_NOT_SAVE_LICENSE, ex);
      return false;
   }

   return true;
}
```

VB.NET

```
Public Overloads Function RefreshLicense() As Boolean
   Dim lfContent As String = ""
   'initialize the object used for calling the web service method
   Using ws As New XmlLicenseFileService()
      If Not Me.RefreshLicense(ws, lfContent) Then
         If Me.LastError.ExtendedErrorNumber = 5010 Or _
            Me.LastError.ExtendedErrorNumber = 5015 Or _
            Me.LastError.ExtendedErrorNumber = 5016 Or _
            Me.LastError.ExtendedErrorNumber = 5017 Then
            File.Delete("[LicenseFile Path")
         End If
         Return False
      End If
   End Using

   'try to save the license file to the file system
   Try
      File.WriteAllText("[License File Path]", lfContent)
   Catch ex As Exception
      Me.LastError = New LicenseError(LicenseError.ERROR_COULD_NOT_SAVE_LICENSE, ex)
      Return False
   End Try

   Return True
End Function
```

Note how this example code deletes the license when SOLO Server returns certain **result codes**. This is because the license should be considered invalid when the application receives one of these result codes from SOLO Server. Deleting the license file is an appropriate response to these result codes for read-only `License` implementations. However, when using `WritableLicense` implementations, you should alter the local license file in-code as appropriate for your licensing requirements, and save it and its aliases.

Refreshing Periodically

Refreshing the license file requires much more server and application resources, so it is important to avoid using this feature too frequently in your protected applications. The **license file** contains a `SignatureDate` property, which tells you when the license file was last issued by SOLO Server. This field can be leveraged to determine the last time the license was validated and refreshed with SOLO Server, and allows you to implement logic to only attempt and/or require your application to phone-home after a certain amount of time has passed. All of the Protection PLUS 5 SDK samples include settings or properties which you may reference and re-configure as needed for your application.

PLUSManaged: Deactivating and Transferring Licenses

Deactivation can occur one of two ways: you may allow customers to deactivate from your application, or you can remotely **deactivate a license or an activation using SOLO Server's administrative interface**.

Important

It is very important to rely on background checking and refreshing when allowing users to deactivate your application! Automated background checking helps prevent users from licensing more systems than intended by restoring the entire system/computer from an image or snapshot taken before deactivation occurred.

In either case, when your application deactivates its license, or runs a **background check or refreshes its license** and detects that it has been deactivated, it can then disable access to the application or certain features.

Important

Be mindful about whether or not disabling access to your application is appropriate. For example, if your application requires high availability (such as a web application or service) and/or your application serves some mission-critical function, it may be more appropriate to have your application display nag messages and/or disable only non-critical features so as to avoid disrupting your customers' business or operation.

Deactivating and Transferring Licenses in your Application

Allowing your customers to deactivate individual activations online is a convenience to them that also reduces your support overhead. Since deactivating allows users to **activate** a new system, this is a very simple and convenient way for your customers to essentially migrate (or transfer) the license from one system to another. An example of when a customer would want to use this functionality would be when he or she is upgrading from one computer to another.

C#

```
public bool DeactivateOnline()
{
  //initialize the object used for calling the web service method
  using (XmlActivationService ws = new XmlActivationService())
  {
     if (DeactivateInstallation(ws))
     {
        File.Delete("[License File Path]");
        return true;
     }
  }

  return false;
}
```

VB.NET

```
Public Function DeactivateOnline() As Boolean
   'initialize the object used for calling the web service method
   Using ws As New XmlActivationService()
      If DeactivateInstallation(ws) Then
         File.Delete("[License File Path]")
         Return True
      End If

      Return False
   End Using
End Function
```

Note how this example code deletes the license when the deactivation is successful. This is because the license should be considered invalid when the application has been deactivated. Deleting the license file is an appropriate response to these result codes for read-only `License` implementations. However, when using `WritableLicense` implementations, you should alter the local license file in-code as appropriate for your licensing requirements, and save it and its aliases.

PLUSManaged: Reading License Fields

Your application will typically need to load **license files** in several areas of your application. This is necessary in order to gain access to the data in the license file, which is typically **encrypted**.

Loading a License File

From Memory

If your application stores the license file in a location other than the file system (such as in your application's database), or if your application has just completed activation or a license refresh, you will need to load the license file from memory. Your **license implementation class** will inherit a `Load` method from `License` or `WritableLicense`, which is the method used to load a license file from memory.

From the File System

If your application stores the license file in a typical file system location (such as in the same folder as your application), then it will need to load the license file from that location during application start-up (so that it may then validate the license). Your **license implementation class** will inherit a `LoadFile` method from `License` or `WritableLicense`, which is the method used to load a license file from the file system.

From an Image File

Important

The steganography feature is presently a beta, and changes may be applied which could require minor source code modifications to be made in order to use a future release. Please keep in mind that **your testing** should be very thorough. Also, use your own images distributed with your application such as a splash screen image. Do not use an existing image such as the desktop wallpaper since another product could share that license location.

If your application stores the license file in a **supported image format** using the Protection PLUS 5 SDK **steganography** feature, then it will need to load the license file from the image location during application start-up (so that it may then validate the license). This is the same as loading from the file system but using an existing image file. Your **license implementation class** will inherit a `LoadFile` method from `License` or `WritableLicense`, which is the method used to load a license file from the image.

Reading License File Data

Once your application has loaded the license file, your **license implementation class** will inherit a wide variety of properties which give you access to all of the license data. This includes information such as the license's effective start and end dates, authorized system identifiers, customer data, and much more (as outlined in the **license file schema**). Your application can then use this data for reacting to the status of the license (such as the expiration/effective end date, or toggling the availability of application features), **copy protection**, displaying details to your users (such as showing the licensee in your application's about dialog), and much more. Additionally, it is important to note that your application may not alter this information directly unless it is **using a writable license**.

PLUSManaged: Using Self-Signed Writable License Files

Some environments, including ones like remote locations (such as in a field or on the road), do not offer any Internet connectivity. Meanwhile, others are highly restrictive with Internet connectivity (often places such as hospitals, government and military offices, etc...). In these types of environments, it can be difficult or impractical to require Internet connectivity. Additionally, there might be other reasons why your application would need to be able to write to its license files freely as well (depending on your **licensing requirements**). In these situations, it is possible to use writable license files, which are license files that are encrypted and signed by your application (**read more about license files**). Although this means all key data needed to write license files is known to your application (which is less secure than using read-only licenses issued by SOLO Server), this affords you extra flexibility when needed.

Keep in mind that you need to make the best choices about what type of license file your application should use, and when it should use it. Our **sample applications** show you how you can use read-only licenses, writable licenses, and even a hybrid of both.

Inheriting from PLUSManaged

When **creating your license implementation class**, the one major thing you need to do differently is to inherit from the `WritableLicense` class (instead of `License`, which is for read-only licenses signed by SOLO Server). Your class would be defined using code similar to that which is shown below.

C#

```
public class SampleLicense : WritableLicense
{
}
```

VB.NET

```
Public Class SampleLicense
   Inherits WritableLicense
End Class
```

Using License File Aliases

Aliases are hidden copies of your application's **license file**, which are designed to prevent your license file from being arbitrarily written to or deleted. Using aliases is necessary when using writable license files, as this prevents users from being able to easily restore the license file to a prior state simply by restoring backup copy of the file.

Important

Although it is critical to use license file aliases to protect your writable licenses, they cannot prevent your application from being returned to a prior state when the entire system is restored to a snapshot or image (using something like Norton Ghost, for example). Using periodic background checking with SOLO Server whenever possible is best for preventing the license from being compromised in this manner.

Configuring Aliases

The first step is to add some code to your license implementation class constructor. The code added will vary depending on what locations you use for your aliases.

Important

The example code below and in any sample application code shows locations that are only meant to provide some guidance on selecting your application's alias locations. Your application must use different alias names and/or locations than any of the examples given by this documentation and any of the sample applications.

Additionally, it is equally important that any two applications (which do not share a license) use separate license file and alias locations.

User-Specific Locations

User-specific alias and license file locations are convenient because they can help you to avoid problems that need to be addressed when users with limited access to a system try to use your software. More specifically, if your application were to use user-specific locations exclusively, the primary benefit would be that users would never need elevated access to run and activate your application. However, the significant drawback is that because the locations used are unique to each user which signs-on to a system, each user will have his or her own license file and aliases, meaning each user will need to activate to be able to use the application. Some example code is provided below to show how you can configure some user-specific alias locations.

Important

The steganography feature is presently a beta, and changes may be applied which could require minor source code modifications to be made in order to use a future release. Please keep in mind that **your testing** should be very thorough. Also, use your own images distributed with your application such as a splash screen image. Do not use an existing image such as the desktop wallpaper since another product could share that license location.

C#

```
//Add an alias in the user's %APPDATA% folder.
this.AddAlias(new LicenseFileSystemAlias(Path.Combine(Path.Combine(Environment.GetFolderPath
(Environment.SpecialFolder.LocalApplicationData), "XyzProduct"), "XyzProductUserAlias1.xml"),
encryptionKey, true));
//Add an alias in the user's registry area.
this.AddAlias(new LicenseWindowsRegistryAlias("Software\\XyzProduct\\License", encryptionKey,
true, Microsoft.Win32.RegistryHive.CurrentUser, "XyzProductUserAlias2"));
//Add an alias to the user's classes\clsid area. Generate and use a new GUID (or a value
formatted like "f8e46dfc-2500-45ba-bc1a-760e0b386147") manually using System.Guid.NewGuid
().ToString(). It is important to use your own GUID instead of the example provided below!
this.AddAlias(new LicenseWindowsRegistryAlias("Software\\Classes\\CLSID\\{f8e46dfc-2500-45ba-
bc1a-760e0b386147}\\Programmable", encryptionKey, true, Microsoft.Win32.RegistryHive.Cur-
rentUser, "Version"));
//Add an alias within an existing Bitmap image located in the user's %APPDATA% folder.
this.AddAlias(new LicenseImageAlias(Path.Combine(Path.Combine(Environment.GetFolderPath(Envir-
onment.SpecialFolder.LocalApplicationData), "XyzProduct"), "Image.bmp"), encryptionKey,
true));
```

VB.NET

```
'Add an alias in the user's %APPDATA% folder.
Me.AddAlias(New LicenseFileSystemAlias(Path.Combine(Path.Combine(Environment.GetFolderPath
(Environment.SpecialFolder.LocalApplicationData), "XyzProduct"), "XyzProductUserAlias1.xml"),
encryptionKey, True))
'Add an alias in the user's registry area.
Me.AddAlias(New LicenseWindowsRegistryAlias("Software\\XyzProduct\\License", encryptionKey,
True, Microsoft.Win32.RegistryHive.CurrentUser, "XyzProductUserAlias2"))
'Add an alias to the user's classes\clsid area. Generate and use a new GUID (or a value
```

```
formatted like "f8e46dfc-2500-45ba-bc1a-760e0b386147") manually using System.Guid.NewGuid
().ToString(). It is important to use your own GUID instead of the example provided below!
Me.AddAlias(New LicenseWindowsRegistryAlias("Software\Classes\CLSID\{f8e46dfc-2500-45ba-bc1a-
760e0b386147}\Programmable", encryptionKey, true, Microsoft.Win32.RegistryHive.CurrentUser,
"Version"))
'Add an alias within an existing Bitmap image located in the user's %APPDATA% folder.
Me.AddAlias(New LicenseImageAlias(Path.Combine(Path.Combine(Environment.GetFolderPath(Envir-
onment.SpecialFolder.LocalApplicationData), "XyzProduct"), "Image.bmp"), encryptionKey,
True))
```

Global Locations

Important

Microsoft Office applications (such as Word, Excel, Access, etc...) may run in a **ClickToRun** environment. This environment has known limitations that make it problematic for licensed Office add-ins and macros to use global locations. Consequently, licensed add-ins and macros that target these environments should only use user-specific locations for licenses and aliases. See **this knowledge-base article** for more details.

Global alias and license file locations are locations which are shared by all users on a system. The advantage to using global locations is that the application only needs to be activated once on a given system. However, the drawback to this is that you typically need to consider setting permissions on these locations when **deploying your application**. Some examples on how you can configure some global alias locations is provided in the code below.

Important

The steganography feature is presently a beta, and changes may be applied which could require minor source code modifications to be made in order to use a future release. Please keep in mind that **your testing** should be very thorough. Also, use your own images distributed with your application such as a splash screen image. Do not use an existing image such as the desktop wallpaper since another product could share that license location.

C#

```
//Add an alias file in the same folder as this application/.exe file.
this.AddAlias(new LicenseFileSystemAlias(Path.Combine(Path.GetDirectoryName(Application.Ex-
ecutablePath), "XyzProductAlias1.xml"), encryptionKey, true));
//Add an alias in the common application data folder (usually "C:\Program Data").
this.AddAlias(new LicenseFileSystemAlias(Path.Combine(Environment.GetFolderPath
(Environment.SpecialFolder.CommonApplicationData), "XyzProductAlias2.xml"), encryptionKey,
true));
//Add an alias in the system folder (usually C:\Windows\System32).
this.AddAlias(new LicenseFileSystemAlias(Path.Combine(Environment.GetFolderPath
(Environment.SpecialFolder.System), "XyzProductAlias3.xml"), encryptionKey, true));
//Add an alias in a global classes\clsid registry area. Generate and use a new GUID (or a
value formatted like "f8e46dfc-2500-45ba-bc1a-760e0b386147") manually using Sys-
tem.Guid.NewGuid().ToString(). It is important to use your own GUID instead of the example
provided below!
this.AddAlias(new LicenseWindowsRegistryAlias("CLSID\\{f8e46dfc-2500-45ba-bc1a-
760e0b386147}\\Programmable", encryptionKey, true, Microsoft.Win32.RegistryHive.ClassesRoot,
"Version"));
//Add an alias file within an existing Bitmap image in the same folder as this applic-
ation/.exe file.
```

```
this.AddAlias(new LicenseImageAlias(Path.Combine(Path.GetDirectoryName(Application.Ex-
ecutablePath), "Image.bmp"), encryptionKey, true));
```

VB.NET

```
'Add an alias file in the same folder as this application/.exe file.
Me.AddAlias(New LicenseFileSystemAlias(Path.Combine(Path.GetDirectoryName(Application.Ex-
ecutablePath), "XyzProductGlobalAlias1.xml"), encryptionKey, True))
'Add an alias in the common application data folder (usually "C:\Program Data").
Me.AddAlias(New LicenseFileSystemAlias(Path.Combine(Environment.GetFolderPath(Environment.Spe-
cialFolder.CommonApplicationData), "XyzProductGlobalAlias2.xml"), encryptionKey, True))
'Add an alias in the system folder (usually C:\Windows\System32).
Me.AddAlias(New LicenseFileSystemAlias(Path.Combine(Environment.GetFolderPath(Environment.Spe-
cialFolder.System), "XyzProductGlobalAlias3.xml"), encryptionKey, True))
'Add an alias in a global classes\clsid registry area. Generate and use a new GUID (or a
value formatted like "f8e46dfc-2500-45ba-bc1a-760e0b386147") manually using Sys-
tem.Guid.NewGuid().ToString(). It is important to use your own GUID instead of the example
provided below!
Me.AddAlias(New LicenseWindowsRegistryAlias("CLSID\{f8e46dfc-2500-45ba-bc1a-760e0b386147}\Pro-
grammable", encryptionKey, True, Microsoft.Win32.RegistryHive.ClassesRoot, "Version"))
'Add an alias file within an existing Bitmap image in the same folder as this applic-
ation/.exe file.
Me.AddAlias(New LicenseImageAlias(Path.Combine(Path.GetDirectoryName(Application.Ex-
ecutablePath), "Image.bmp"), encryptionKey, True))
```

Validating Aliases

Adding logic to your application to validate its configured aliases is pivotal to using aliases properly. The code below illustrates how you can validate aliases in your license implementation class.

C#

```
LicenseAliasValidation aliasValidation = new LicenseAliasValidation(this);
if (!aliasValidation.Validate())
{
   this.LastError = aliasValidation.LastError;
   return false;
}
```

VB.NET

```
Dim aliasValidation As New LicenseAliasValidation()
If Not aliasValidation.Validate() Then
   Me.LastError = aliasValidation.LastError
   Return False
End If
```

Important

Your license validation logic also needs to verify the system clock when using aliases to help ensure you can trust the time the system clock reports to your application.

Self-Issued Evaluations

Evaluations are one of the most common license models, and it is often convenient (if not necessary) to allow your application to automatically issue evaluation periods for your prospective customers (**learn more**).

Identifying a Self-Issued Evaluation

If your application uses a writable license for all types of licenses, then it needs to be able to distinguish evaluations from other types of licenses. Since all other types of licenses will generally require activation with SOLO Server, the easiest way to identify self-issued evaluations is to check for an empty string value in the `InstallationID` property (as an Installation ID would be present for any licenses activated with SOLO Server). Otherwise, if your application only uses writable licenses for evaluations, then this can easily be determined by which license implementation is being used.

Creating Fresh Evaluations

When your application is run on a computer for the first time, it is common practice to automatically issue a fresh evaluation period. You can implement this functionality in your license implementation class using a method similar to the one shown below.

C#

```
public bool CreateFreshEvaluation()
{
   //Start by loading and checking all of the aliases.
   int numAliases, numValidAliases;
   this.CheckAliases(out numAliases, out numValidAliases);

   //If we found any aliases, write the most recent one as the license file, and use it
   instead of creating a fresh evaluation.
   LicenseAlias mostRecent = LicenseAlias.GetMostCurrentAlias(this.Aliases);
   if (mostRecent.LastUpdated != DateTime.MinValue)
   {
      //Write the most recent alias as the new, primary license file and load its contents.
      this.WriteAliasToLicenseFile(mostRecent, "[License File Path]");
      this.Load(mostRecent.Contents);

      //Now update all the other aliases as well.
      int aliasesToWrite, aliasesWritten;
      this.WriteAliases(out aliasesToWrite, out aliasesWritten);

      return true;
   }

   //Since evaluations are not activated, they have no Installation ID.
   this.InstallationID = "";
   //Set the license's effective start date to the current date.
   this.EffectiveStartDate = DateTime.UtcNow.Date;
   //Set the license's effective end date (the expiration date) to 30 days after the current
   date.
   this.EffectiveEndDate = DateTime.UtcNow.Date.AddDays(30);

   int filesToWrite, filesWritten;
   this.WriteAliases(out filesToWrite, out filesWritten);
```

```
   //TODO: you can add your own logic here to set your own requirements for how many aliases
   must be written
   // ...for this example, we only require 1.
   if (filesWritten < 1)
   {
      return false;
   }

   return this.WriteLicenseFile("[License File Path]");
}
```

VB.NET

```
Public Function CreateFreshEvaluation() As Boolean
   'Start by loading and checking all the aliases.
   Dim numAliases As Integer, numValidAliases As Integer
   Me.CheckAliases(numAliases, numValidAliases)

   'If we found any aliases, write the most recent one as the license file, and use it
   instead of creating a fresh evaluation.
   Dim mostRecent As LicenseAlias = LicenseAlias.GetMostCurrentAlias(Me.Aliases)
   If mostRecent.LastUpdated <> DateTime.MinValue Then
      Me.WriteAliasToLicenseFile(mostRecent, "[License File Path]")
      'Write the most recent alias as the new, primary license file and load its contents.
      Me.Load(mostRecent.Contents)

      'Now update all the other aliases as well.
      Dim aliasesToWrite As Integer, aliasesWritten As Integer
      Me.WriteAliases(aliasesToWrite, aliasesWritten)

      Return True
   End If

   'Since evaluations are not activated, they have no Installation ID.
   Me.InstallationID = ""
   'Set the license's effective start date to the current date.
   Me.EffectiveStartDate = DateTime.Now.[Date]
   'Set the license's effective end date (the expiration date) to 30 days after the current
   date.
   Me.EffectiveEndDate = DateTime.Now.[Date].AddDays(30)

   Dim filesToWrite As Integer, filesWritten As Integer
   Me.WriteAliases(filesToWrite, filesWritten)

   'TODO: you can add your own logic here to set your own requirements for how many aliases
   must be written
   ' ...for this example, we only require 1.
   If filesWritten < 1 Then
      Return False
   End If

   Return Me.WriteLicenseFile("[License File Path]")
End Function
```

Then, you can update your application to check to see whether or not the license file exists. When no license file is present (or when the `LoadFile` method inherited from `WritableLicense` fails), you can then call the new `CreateFreshEvaluation` method.

Creating Expired Evaluations

If your application uses writable license files exclusively (and not a hybrid of writable and read-only license files), then there may be some cases where you need to create an expired evaluation. This can include situations such as when a customer deactivates a license that was previously activated, or when a background check finds that the License ID or the Installation ID is no longer valid (or has been deactivated or revoked). Below is an example showing how you can create a method that creates an expired evaluation license, which you can then have your application call under these kinds of circumstances.

C#

```
public bool CreateExpiredEvaluation()
{
   //Since evaluations are not activated, they have no Installation ID.
   this.InstallationID = "";
   //Set the license's effective start date to yesterday.
   this.EffectiveStartDate = DateTime.UtcNow.Date.AddDays(-1);
   //Set the license's effective start date to yesterday so it is expired.
   this.EffectiveEndDate = DateTime.UtcNow.Date.AddDays(-1);

   int filesToWrite, filesWritten;
   this.WriteAliases(out filesToWrite, out filesWritten);

   //TODO: you can add your own logic here to set your own requirements for how many aliases
   must be written
   // ...for this example, we only require 1.
   if (filesWritten < 1)
   {
      return false;
   }

   return this.WriteLicenseFile("[License File Path]");
}
```

VB.NET

```
Public Function CreateExpiredEvaluation() As Boolean
   'Since evaluations are not activated, they have no Installation ID.
   Me.InstallationID = ""
   'Set the license's effective start date to yesterday.
   Me.EffectiveStartDate = DateTime.UtcNow.[Date].AddDays(-1)
   'Set the license's effective start date to yesterday so it is expired.
   Me.EffectiveEndDate = DateTime.UtcNow.[Date].AddDays(-1)

   Dim filesToWrite As Integer, filesWritten As Integer
   Me.WriteAliases(filesToWrite, filesWritten)

   'TODO: you can add your own logic here to set your own requirements for how many aliases
   must be written
   ' ...for this example, we only require 1.
   If filesWritten < 1 Then
      Return False
   End If

   Return Me.WriteLicenseFile("[License File Path]")
End Function
```

PLUSManaged: Adding Basic Copy Protection

While **creating your license implementation class**, you should have created a `Validate` method. This is where you can implement your copy protection logic by adding validation of information in the license and the system running your application.

Adding System Identifier Validation

System identification is the primary means of preventing a license from working on a system other than the one on which it was activated. In other words, if you were to activate an application on Computer A, copy Computer A's license file to Computer B, the application would prevent the application from running on Computer B using Computer A's license file.

Note

More detailed information on each system identifier can be found in the PLUSManaged API documentation under the **com.softwarekey.Client.Licensing** namespace.

Basic System Identifier Validation

To add basic system identifier validation, we can update the `Validate` method added above to look like the following:

C#

```
public bool Validate()
{
   SystemIdentifierValidation systemIdValidation = new SystemIdentifierValidation(this.Author-
   izedIdentifiers,
      this.CurrentIdentifiers,
      SystemIdentifierValidation.REQUIRE_EXACT_MATCH);
   if (!systemIdValidation.Validate())
   {
      this.LastError = systemIdValidation.LastError;
      return false;
   }

   return true;
}
```

VB.NET

```
Public Function Validate() As Boolean
   Dim systemIdValidation As New SystemIdentifierValidation(Me.AuthorizedIdentifiers, _
      Me.CurrentIdentifiers, _
      SystemIdentifierValidation.REQUIRE_EXACT_MATCH)
   If Not systemIdValidation.Validate() Then
      Me.LastError = systemIdValidation.LastError
      Return False
   End If

   Return True
End Function
```

The code above checks to make sure the system's current identifiers are an exact match to the identifiers authorized during activation. If the `CurrentIdentifiers` do not match the `AuthorizedIdentifiers` in the license file exactly,

then the license is rejected by the example above. If you only use a single type of SystemIdentifierAlgorithm, it is possible for you to replace SystemIdentifierValidation.REQUIRE_EXACT_MATCH to a number representing the minimum number of identifiers that must match instead. However, if you use several different SystemIdentifierAlgorithm implementations in your application, then you should consider adding your own, customized validation.

Customized Validation and Fuzzy-Matching

It is possible to customize your validation logic to allow for some amount of acceptable change/variation by comparing the hash values of certain types of CurrentIdentifiers against the hash values of the same type of AuthorizedIdentifiers. To implement this in your application, you can add a new method to your license implementation class, which validates the identifiers using custom logic similar to what is shown below.

Important

Do not use the SystemIdentifierValidation with customized identifier validation and matching, as the two will likely conflict with each other.

C#

```
private bool ValidateIdentifiers()
{
   int formatSerials = 0;
   int formatSerialMatches = 0;
   int nics = 0;
   int nicMatches = 0;
   int computerNames = 0;
   int computerNameMatches = 0;

   //calculate the number of authorized identifiers of each type, and calculate how many
   matches are found
   foreach (SystemIdentifier authorizedIdentifier in this.AuthorizedIdentifiers)
   {
      SystemIdentifier matchingIdentifier = null;

      foreach (SystemIdentifier currentIdentifier in this.CurrentIdentifiers)
      {
         //Use the SystemIdentifier class's == operator overload to compare the value hash and
         type
         if (currentIdentifier == authorizedIdentifier)
         {
            //we found a match
            matchingIdentifier = currentIdentifier;
            break;
         }
      }

      //update our counters
      switch (authorizedIdentifier.Type)
      {
         case "NicIdentifier":
            nics++;
            if (matchingIdentifier != null)
               nicMatches++;
            break;
         case
```

```
            "HardDiskVolumeSerialIdentifier":
               formatSerials++;
               if (matchingIdentifier != null)
                  formatSerialMatches++;
               break;
            case "ComputerNameIdentifier":
               computerNames++;
               if (matchingIdentifier != null)
                  computerNameMatches++;
               break;
         }
      }

      //Make sure we have at least one match for each type of identifier we have authorized
      if ((formatSerialMatches < 1) ||
         (nicMatches < 1) ||
         (computerNameMatches < 1))
      {
         LastError = new LicenseError(LicenseError.ERROR_LICENSE_SYSTEM_IDENTIFIERS_DONT_MATCH);
         return false;
      }

      return true;
   }
```

VB.NET

```
Private Function ValidateIdentifiers() As Boolean
   Dim formatSerials As Integer = 0
   Dim formatSerialMatches As Integer = 0
   Dim nics As Integer = 0
   Dim nicMatches As Integer = 0
   Dim computerNames As Integer = 0
   Dim computerNameMatches As Integer = 0

   'calculate the number of authorized identifiers of each type, and calculate how many
   matches are found
   For Each authorizedIdentifier As SystemIdentifier In Me.AuthorizedIdentifiers
      Dim matchingIdentifier As SystemIdentifier = Nothing

      For Each currentIdentifier As SystemIdentifier In Me.CurrentIdentifiers
         'Use the SystemIdentifier class's Equals override method to compare the value hash
         and type
         If currentIdentifier.Equals(authorizedIdentifier) Then
            'we found a match
            matchingIdentifier = currentIdentifier
            Exit For
         End If
      Next

      'update our counters
      Select authorizedIdentifier.Type
         Case "NicIdentifier"
            nics += 1
            If Not matchingIdentifier Is Nothing Then nicMatches += 1
         Case "HardDiskVolumeSerialIdentifier"
            formatSerials += 1
            If Not matchingIdentifier Is Nothing Then formatSerialMatches += 1
         Case "ComputerNameIdentifier"
            computerNames += 1
            If Not matchingIdentifier Is Nothing Then computerNameMatches += 1
      End Select
   Next

   'Make sure we have at least one match for each type of identifier we have authorized
   If (formatSerialMatches < 1) Or _
      (nicMatches < 1) Or _
      (computerNameMatches < 1) Then
      LastError = New LicenseError(LicenseError.ERROR_LICENSE_SYSTEM_IDENTIFIERS_DONT_MATCH)
      Return False
   End If

   Return True
End Function
```

The example code above assumes three types of `SystemIdentifierAlgorithm` are used by the application, and requires at least one match for each. Once your customized validation method is implemented, you can update your Validate method to call to it as shown below.

C#

```
public bool Validate()
{
   if (!this.ValidateIdentifiers())
   {
      this.LastError = systemIdValidation.LastError;
      return false;
   }

   return true;
}
```

VB.NET

```
Public Function Validate() As Boolean
   If Not Me.ValidateIdentifiers() Then
      Me.LastError = systemIdValidation.LastError
      Return False
   End If

   Return True
End Function
```

PLUSManaged: Validating Time-Limited and Subscription Licenses

Validating time-limited licenses is simple in principle. To do this you can simply check to make sure the system's current date (`DateTime.UtcNow.Date`) is not past the license file's `EffectiveEndDate`. However, the tricky part is checking to make sure your application can trust the licensed system is reporting a reasonably accurate date. PLUSManaged offers a variety of ways to validate the system's date, and it is up to you to pick and chose the methods most appropriate and reliable for your application based upon its needs and the expectations of the environments in which the application will run. If your licensing requirements never impose any time limits, then you may not need to worry about validating the licensed system's clock (although it would still be good to read through and understand the subject for future reference).

Local Validation

Local validations are validations or checks your application can perform to prevent system clock tampering. PLUSManaged offers a number of pieces of information and validation objects that you can use.

API Tampering

Regardless of whether or not .NET is being used, your application will ultimately end up making calls to Windows API functions to get the system's date and time. There are some tools available (such as "Time Stopper" or "RunAsDate"), which essentially take the place of these Windows API functions in your application (via function hooking) and send an altered date to your application. Unfortunately, this type of hack is very impractical to prevent without being very invasive. However, the good news is that detecting this type of hack is generally very easy, as these types of tools generally result in the underlying APIs always returning the same time. PLUSManaged makes this very easy to detect and react to by adding code that uses the `SystemClockValidation` to your license implementation's `Validate` method as shown in the example below.

C#

```
SystemClockValidation clockValidation = new SystemClockValidation();
if (!clockValidation.Validate())
{
   this.LastError = clockValidation.LastError;
   return false;
}
```

VB.NET

```
Dim clockValidation As New SystemClockValidation()
If Not clockValidation.Validate() Then
   Me.LastError = clockValidation.LastError
   Return False
End If
```

Evaluating Date Properties

There are several properties inherited in your license implementation class that you can evaluate to prevent system clock tampering. These properties include:

- `EffectiveStartDate` is the date and time which the license was issued. If the system clock predate the value stored here, the application should treat the license as being invalid or expired.
- `EffectiveEndDate` is the date and time which the license expires. Any time-limited license should only be treated as a valid license when the system clock predates or equals this property's value. If the system clock shows a date and time that occurs after the date stored in this property, the application should treat the license as being expired.

- `SignatureDate` is the date and time which SOLO Server signed the license file which was loaded by the application. If the system clock predates the value stored in this property's value, the application should treat the license as being invalid or expired.
- `LastUpdated` is the date and time which a writable license was written, and can therefore only be used with writable license files. If the system clock predates the value stored in this property's value, the application should treat the license as being invalid or expired.

Enforcing Time Limitations

When using a time-limited license, PLUSManaged includes a `LicenseEffectiveDateValidation` class that your application can leverage to validate the `EffectiveStartDate`, `EffectiveEndDate`, and `SignatureDate` properties. The example below shows how your application can validate these dates (and this code excerpt assumes it is in a method in your license implementation class, as the keyword `this`/`Me` represents a `License` object).

C#

```
LicenseEffectiveDateValidation dateValidation = new LicenseEffectiveDateValidation(this);
if (!dateValidation.Validate())
{
   this.LastError = dateValidation.LastError;
   return false;
}
```

VB.NET

```
Dim dateValidation As New LicenseEffectiveDateValidation(Me)
If Not dateValidation.Validate() Then
   Me.LastError = dateValidation.LastError
   Return False
End If
```

Alias Validation

When using a writable license (regardless of whether or not it is time-limited), it is always best to validate the `LastUpdated` property on the license file and aliases to ensure a backup of the license file has not been restored, and that the system clock has not been back-dated. To help simplify the process of validating the aliases, PLUSManaged includes a `LicenseAliasValidation` that your application may use when validating the license and/or its aliases. An example of how to use this class in your license validation class is illustrated below.

C#

```
LicenseAliasValidation aliasValidation = new LicenseAliasValidation(this);
if (!aliasValidation.Validate())
{
   this.LastError = aliasValidation.LastError;
   return false;
}
```

VB.NET

```
Dim aliasValidation As New LicenseAliasValidation(Me)
If Not aliasValidation.Validate() Then
   Me.LastError = aliasValidation.LastError
   Return False
End If
```

Internet Validation

When possible, using an Internet source to validate a system's clock is a good way to determine whether or not the system clock is reporting a reasonably accurate time.

SOLO Server Web Services

When your application submits requests to SOLO Server's web services (including XmlActivationService, XmlLicenseFileService, and XmlNetworkFloatingService), SOLO Server checks the requesting system's date and time. Instant SOLO Server is configured to require the requesting system's date and time to be within 24 hours of the server's date and time. If you have configured SOLO Server or use anInstant SOLO Server Dedicated URL, you have the option **contact us** to obtain additional information on adjusting this requirement. Whenever the requesting system is outside of SOLO Server's requirement, the web service response will reflect an error with a **result code** of 5022.

Network Time Protocol

Network Time Protocol (NTP) is the protocol used by the vast majority of computers to synchronize the system clock with an Internet time server. Although there is some benefit of validating time against an additional external source, using NTP has several disadvantages and challenges, including:

- NTP servers are generally only meant to be used for synchronization, and may not be able to cope with the load imposed by applications validating the system's clock. If you are not hosting your own Internet time server for your application to use, then you should contact the server operator and obtain permission before using any server for validation in your application. It is very important to avoid the same **controversy** from inadvertent misuse of NTP as experienced by others in the past.
- The NTP protocol does not offer any integrity validation, so there is no practical way to ensure the server you intend to use for validation is actually the one which responded to your application's NTP requests.
- Since the NTP protocol is a sessionless user datagram protocol (UDP), it is not reliable as TCP-based protocols. Therefore, it is not uncommon to see attempts to reach out to NTP servers fail, especially when the request originates behind a router, firewall, or proxy server.
- PLUSManaged only supports this type of validation in Windows, using the .NET Framework, and does not support this validation under Mono and other platforms.

If you opt to use NTP validation, it can be implemented very easily with PLUSManaged by adding a call to `AddTimeServerCheck` in your license implementation's constructor, and then calling the `CheckTimeAgainstServers` method in your license implementation's `Validate` method.

Automatic Recurring Payments

SOLO Server's Payment Plans allow you to offer subscription licenses, maintenance and support subscriptions, payment over multiple installments, and more. To learn more about how to configure SOLO Server for Payment Plans, please refer to the **SOLO Server manual**.

PLUSManaged: Identifying Virtual Machine Guest Environments

A good example of a Virtual Machine (VM) guest environment is where the Windows operating system is running in an isolated environment in OS X (using software like VMWare Fusion or Parallels). Although **copy protection** is designed to prevent copying a license from one machine to another, its effectiveness is often limited when the license machine happens to be a virtual machine. This is due to the fact that the virtual hardware (and the device drivers for that hardware) does not change after copying the virtual machine from one host computer to another.

License Requirements and Considerations

The way your application should react to a virtual machine guest environment will largely depend on the nature of your application and the systems on which it is run. For example, if your application is a typical desktop GUI application that should be activated once on each licensed computer, then you might opt to prevent your application from running in this type of environment. However, if your application needs to run in large office environments where employees use virtualized desktop environments, or if your application provides some kind of service (especially one which requires high-availability, such as a web application or web service), you may need to allow the application to run in these types of environments. If your application falls into the latter category, then you should take further consideration into the types of system identifiers used by your application. For example, if your application is a web application or web service, then one option may be to include the fully qualified host and domain name as a system identifier (you can use the `ServerHostNameIdentifierAlgorithm` for this purpose). You might also want to evaluate other pieces of information that might make for suitable, **custom identifiers**.

Detecting Virtual Machine Guest Environments

PLUSManaged makes it very easy to detect virtual machine guest environments via the `VirtualMachineValidation` **class**. The code below provides an example of how this can be used in your **license implementation class** (i.e. in the `Validate` method).

C#

```
VirtualMachineValidation vmValidation = new VirtualMachineValidation();
vmValidation.Validate();
if (vmValidation.IsVirtual)
{
   //TODO: Add code to have your application react to virtual machine guest environments
   here.
}
else
{
   //TODO: If your application needs logic for non-virtual environments, add it here.
}
```

VB.NET

```
Dim vmValidation As New VirtualMachineValidation()
vmValidation.Validate()
If vmValidation.IsVirtual Then
   'TODO: Add code to have your application react to virtual machine guest environments here
Else
   'TODO: If your application needs logic for non-virtual environments, add it here.
End If
```

System Identification on Virtual Machine Guests

As mentioned in the introduction of this topic, **copy protection** is limited in how well it can prevent a protected application from running in a copied virtual machine guest environment. If you need to license virtual machine guest environments regardless of the limitations,there are some reasonable measures you can take to help prevent this sort of thing from happening, which is what is explained here.

Algorithms

PLUSManaged includes a number of `SystemIdentifierAlgorithm` implementations that can help you better identify both physical machines and virtual machine guests. This section will explain what algorithms are most effective for virtual machine guest environments, and the behavior you can expect from each of the algorithms.

BIOS UUID

With physical computers, the `BiosUuidIdentifierAlgorithm` is useful for uniquely identifying a motherboard, and this is also useful for uniquely identifying an specific virtual machine guest. This means that if a user creates two new virtual machine guests, which we will call *Guest A* and *Guest B*, and installs any operating system on them, they will receive different identifiers for each unique guest. However, if the user were to make a copy of either guest, you might not see a different UUID for that copy of the guest. If you are using something like VMWare ESX Server, you should see a unique UUID even for the copy, as it requires a unique UUID to manage each individual guest it hosts.

Hard Disk Volume Serial

The identifiers generated with the `HardDiskVolumeSerialIdentifierAlgorithm` use data that is generated when a partition is formatted. Since this is usually done when installing the operating system, this value is usually different between different virtual machine guests. However, as with physical machines, use of system imaging software (such as Backup Exec System Recovery) can duplicate an operating system installation, including the volume's serial number. With this limitation in mind, this algorithm is best used when accompanied by another algorithm such as the `BiosUuidIdentifierAlgorithm`.

Network Interface Card

The MAC (or physical) address of Network Interface Cards (NICs), which may be identified with the `NicIdentifierAlgorithm`, is also useful for identifying both physical machines and virtual machine guests. This is because two machines cannot have connectivity on the same network when they share the same MAC address. While this may be useful, it should be noted that most hypervisors allow you to configure the virtual machine guests to use Network Address Translation (NAT), which essentially hides the virtual machine guest behind a separate network. In other words, two virtual machines sharing the same MAC address with bridged connections will not work on the same network, but they may work on separate hosts when using NAT.

Processor

The `ProcessorIdentiferAlgorithm` provides information about the processor's name, vendor/manufacturer, and version (which includes the model and stepping). In virtual machine guest environments, this information about the processor generally matches the host (although it is sometimes possible for further isolation to be used, which is uncommon since it usually has a negative impact on performance and features for the virtual machine guest). Using this information can help you prevent scenarios where a user copies a virtual machine guest created with VMWare Workstation on a PC to a Mac host running VMWare Fusion, for example. However, if your application needs to allow the guest to float to different hosts (i.e. when using VMWare ESX with VMotion), then this data should change, and is counter-productive to use for uniquely identifying the licensed system.

PLUSManaged: Custom SystemIdentifierAlgorithm Implementations

The PLUSManaged library is designed so that it is extremely extensible to allow for highly customized implementations. This includes a framework which enables you to implement your own means of uniquely identifying what is authorized to use a given license. The `SystemIdentifierAlgorithm` and `SystemIdentifier` abstract classes may be derived to implement your algorithm, and this walk-through explains how to do this in detail. To get the most out of this walk-through, we suggest making a copy of one of the **sample applications** and following the steps outlined below in that new copy.

Important

It is imperative to keep in mind that you are responsible for testing any custom system identifier algorithms you create to ensure that the reliability and uniqueness of the identifiers meet your expectations and requirements.

Before you begin, you will want to simply determine the name of your algorithm. For the purposes of this walk-through, we will simply call this "CustomExample".

Implementation

Creating the Classes

First, we will create our two new classes required to implement our "CustomExample" algorithm: `CustomExampleIdentifierAlgorithm` and `CustomExampleIdentifier`. These new classes will:

1. Be defined in files which match the class name ("CustomExampleIdentifierAlgorithm.cs" and "CustomExampleIdentifier.cs", respectively)
2. Extend the `SystemIdentifierAlgorithm` and `SystemIdentifier` classes in PLUSManaged, respectively.
3. Reference the `com.softwarekey.Client.Licensing` namespace.

Following the instructions above, the `ExampleCustomIdentifier` class should initially look something like this:

C#

```
using System;
using System.Collections.Generic;
using com.softwarekey.Client.Licensing;

namespace YourApplicationsNamespace
{
   public class ExampleCustomIdentifier : SystemIdentifier
   {
   }
}
```

Visual Basic .NET

```
Imports System
Imports System.Collections.Generic
Imports com.softwarekey.Client.Licensing

Namespace YourApplicationsNamespace
   Public Class ExampleCustomIdentifier
      Inherits SystemIdentifier
   End Class
End Namespace
```

...And the `ExampleCustomIdentifierAlgorithm` class should initially look something like this:

C#

```
using System;
using System.Collections.Generic;
using com.softwarekey.Client.Licensing;

namespace YourApplicationsNamespace
{
   public class ExampleCustomIdentifierAlgorithm : SystemIdentifierAlgorithm
   {
   }
}
```

Visual Basic .NET

```
Imports System
Imports System.Collections.Generic
Imports com.softwarekey.Client.Licensing

Namespace YourApplicationsNamespace
   Public Class ExampleCustomIdentifierAlgorithm
      Inherits SystemIdentifierAlgorithm
   End Class
End Namespace
```

Implementing the Custom Identifier Class

Now that the identifier class is defined, the next step is to define and implement the required members and constructors...

1. Add a `protected static int` member variable, which will be used to automatically increment default ID's for the collection of identifiers. In this example, we named this `m_customExampleIdentifierId` and defaulted its value to 1.
2. Several constructors, which all need to call base class constructors, need to be created...
   1. A default constructor that does not accept any arguments.
   2. A constructor that accepts an identifier's value.
   3. A constructor that accepts an identifier's name and value
   4. A constructor that accepts an identifier's name, value, and a hash.
3. Next, the constructors need to be implemented...
   1. Constructors 1 and 2 above are meant to be used when generating a new list of identifiers for the current system, so these constructors need to use the static integer to pre-populate the name and increment the static integer. Since the second constructor also receives a value, it must also pre-populate the member variable used for the value property.
   2. The last two constructors are meant for loading authorized identifiers from a license file, so these do not need to pre-populate the name.
4. Finally, the class also needs to override the Type property so it returns the name of the class.

Following the steps above, the `ExampleCustomIdentifier` class should now look something like this:

C#

```
using System;
using System.Collections.Generic;
using com.softwarekey.Client.Licensing;
```

```
namespace YourApplicationsNamespace
{
   public class ExampleCustomIdentifier : SystemIdentifier
   {
      protected static int m_exampleCustomIdentifierId = 1;

      public ExampleCustomIdentifier()
      {
         m_name = "ExampleCustomIdentifier" + m_exampleCustomIdentifierId.ToString();
         m_exampleCustomIdentifierId++;
      }

      public ExampleCustomIdentifier(string value)
      {
         m_name = "ExampleCustomIdentifier" + m_exampleCustomIdentifierId.ToString();
         m_exampleCustomIdentifierId++;

         m_value = value;
      }

      public ExampleCustomIdentifier(string name, string value)
         : base(name, value)
      {
      }

      public ExampleCustomIdentifier(string name, string value, string hash)
         : base(name, value, hash, "ExampleCustomIdentifier")
      {
      }

      public override string Type
      {
         get { return "ExampleCustomIdentifier"; }
      }
   }
}
```

Visual Basic .NET

```
Imports System
Imports System.Collections.Generic
Imports com.softwarekey.Client.Licensing

Namespace YourApplicationsNamespace
   Public Class ExampleCustomIdentifier
      Inherits SystemIdentifier

      Protected Shared m_exampleCustomIdentifierId As Integer = 1

      Public Sub New()
         m_name = "ExampleCustomIdentifier" & m_exampleCustomIdentifierId.ToString()
         m_exampleCustomIdentifierId += 1
      End Sub

      Public Sub New(ByVal value As String)
         m_name = "ExampleCustomIdentifier" & m_exampleCustomIdentifierId.ToString()
         m_exampleCustomIdentifierId += 1

         m_value = value
      End Sub

      Public Sub New(ByVal name As String, ByVal value As String)
         MyBase.New(name, value)
      End Sub

      Public Sub New(ByVal name As String, ByVal value As String, ByVal hash As String)
         MyBase.New(name, value, hash, "ExampleCustomIdentifier")
      End Sub

      Public Overrides ReadOnly Property Type As String
         Get
            Return "ExampleCustomIdentifier"
         End Get
      End Property
   End Class
End Namespace
```

This concludes the implementation of the `ExampleCustomIdentifier` class.

Implementing the Custom Algorithm Class

With the algorithm class defined, the first step is to implement a default constructor for the class, which also must call the `base` class (`SystemIdentifierAlgorithm`) constructor and pass in the ID of the algorithm (which should really only be the same as the name of the algorithm). So the class file would then look something like this:

C#

```
using System;
using System.Collections.Generic;
using com.softwarekey.Client.Licensing;

namespace YourApplicationsNamespace
```

```
{
   public class ExampleCustomIdentifierAlgorithm : SystemIdentifierAlgorithm
   {
      public ExampleCustomIdentifierAlgorithm() : base("ExampleCustom") { }
   }
}
```

Visual Basic .NET

```
Imports System
Imports System.Collections.Generic
Imports com.softwarekey.Client.Licensing

Namespace YourApplicationsNamespace
   Public Class ExampleCustomIdentifierAlgorithm
      Inherits SystemIdentifierAlgorithm

      Public Sub New()
         MyBase.New("ExampleCustom")
      End Sub
   End Class
End Namespace
```

Of course, you are free to add your own initialization logic in the default constructor here. Since this example does not require any custom initialization logic, the constructor will remain empty and will do nothing more than simply call the base constructor of `SystemIdentifierAlgorithm`.

Lastly, we will need to implement an override for the `GetIdentifiers()` method. This method is used to generate a list of identifiers representing the current system, and is typically where you implement your logic to provide the proper values that are used to identify the authorized system. Since this method returns a generic list, the list returned may return one or more than one identifier, though this example will only use a single identifier. An example of an algorithm that only ever returns a single identifier here is the built-in `ComputerNameIdentifierAlgorithm`. Obviously, this is because a computer may only have a single name at any given time. On the other hand, the `NicIdentifierAlgorithm` can return multiple entries in the list because computers may have multiple Network Interface Cards (NICs). This could occur in a personal computer with built-in wired and wireless network adapters; or it could occur with server that has multiple, redundant network adapters. So following these instructions, the completed class should look something like this:

C#

```
using System;
using System.Collections.Generic;
using com.softwarekey.Client.Licensing;

namespace YourApplicationsNamespace
{
   public class ExampleCustomIdentifierAlgorithm : SystemIdentifierAlgorithm
   {
      public ExampleCustomIdentifierAlgorithm() : base("ExampleCustom") { }

      public override List<SystemIdentifier> GetIdentifiers()
      {
         List<SystemIdentifier> identifiers = new List<SystemIdentifier>();

         identifiers.Add(new ExampleCustomIdentifier("ExampleCustomIdentifier1", "My Example
```

```
        Value"));

        return identifiers;
      }
   }
}
```

Visual Basic .NET

```
Imports System
Imports System.Collections.Generic
Imports com.softwarekey.Client.Licensing

Namespace YourApplicationsNamespace
   Public Class ExampleCustomIdentifierAlgorithm
      Inherits SystemIdentifierAlgorithm

      Public Sub New()
         MyBase.New("ExampleCustom")
      End Sub

      Public Overrides Function GetIdentifiers() As List(Of SystemIdentifier)
         Dim identifiers As New List(Of SystemIdentifier)

         identifiers.Add(New ExampleCustomIdentifier("ExampleCustomIdentifier1", "My Example
         Value"))

         Return identifiers
      End Function
   End Class
End Namespace
```

That concludes the steps required to implement your own custom system identifier algorithm.

Using a Custom Identifier Algorithm

Once your class is implemented, all you need to do to use it is add it to the list of algorithms being used in your license implementation. In the PLUSManaged samples and the PLUSManagedGui, this is defined in the `SystemIdentifierAlgorithms` property in the `LicenseConfiguration` class. This is then passed in to the `License` or `WritableLicense` (or the base class) constructor in your **license implementation class**. Below is a rough example of how you can modify the LicenseConfiguration class included in the samples to use your new custom algorithm.

C#

```
internal static List<SystemIdentifierAlgorithm> SystemIdentifierAlgorithms
{
   get
   {
      return new List<SystemIdentifierAlgorithm>(
         new SystemIdentifierAlgorithm[] {
            new ExampleCustomIdentifierAlgorithm(),
            /*All other algorithms you use go here.*/ });
   }
}
```

Visual Basic .NET

```
Friend Shared ReadOnly Property SystemIdentifierAlgorithms() As List(Of Sys-
temIdentifierAlgorithm)
   Get
      Return New List(Of SystemIdentifierAlgorithm)(New SystemIdentifierAlgorithm() { New
      ExampleCustomIdentifierAlgorithm(), _
         [All other algorithms used go here] })
   End Get
End Property
```

Accepting User-Defined Input for an Algorithm

If you wish to accept user-defined input, you should omit the step described in the "Using a Custom Identifier Algorithm" section just above. First, you will need to prompt the user to enter the data before initializing the `SampleLicense` class. In the sample applications, you would do this in the `MainForm_Load` event handler, just before it loads the license. You would store the data in a private member variable of the `MainForm` class so that it may be re-used when the application reloads the license, and avoid the need to prompt the user each time this might happen. Next, in the `SampleLicense` class's constructor, you will want to manually add the identifier to the list of current identifiers. For this example, the code to accomplish this would look something like:

C#

```
CurrentIdentifiers.Add(new ExampleCustomIdentifier("My Example Value"));
```

Visual Basic .NET

```
CurrentIdentifiers.Add(New ExampleCustomIdentifier("My Example Value"))
```

The hard-coded string, `"My Example Value"`, would instead be replaced by the user-input, which could be passed to the `ExampleCustomIdentifier` class by adding a new constructor argument and public property (with get and set methods implemented).

PLUSManaged: Adding Support for Proxy Servers

Important

If you are using or planning to use **PLUSManagedGui** in your application, this content may not be relevant to you since PLUSManagedGui automatically handles common functionality.

Some environments limit and monitor Internet connectivity using proxy servers (often places such as hospitals, government and military offices, etc...), which can have an impact on your application's ability to activate. This topic helps you to establish compatibility with most proxy servers.

Using the Sample Code

The easiest way to add support to your application is to copy some code from the sample projects included with Protection PLUS 5 SDK. This includes the `ProxyServerCredentialsForm` and the `WebServiceHelper` classes, which help you manage proxy server and proxy authentication credentials in your application. You will want to search the source code for any comments that start with `TODO` to see if there is any code that you should alter before shipping it with your application. At the top of your license implementation class, you can add a static/shared member as follows:

C#

```
private static WebServiceHelper m_WebServiceHelper = new WebServiceHelper();
```

VB.NET

```
Private Shared m_WebServiceHelper As New WebServiceHelper()
```

Then, you can update code which created web service objects directly to use the `WebServiceHelper` methods instead. Using `XmlActivationService` as an example:

C#

```
XmlActivationService ws = m_WebServiceHelper.GetNewXmlActivationServiceObject();
```

VB.NET

```
Dim ws As XmlActivationService = m_WebServiceHelper.GetNewXmlActivationServiceObject()
```

It is possible for the object to fail to initialize properly, so it is important to make sure the object returned is not `null` (C#) or `Nothing` (VB.NET). If you find you did not get an object back, check the `WebServiceHelper` object's `LastError` property for details.

Adding Customized Proxy Server Support

Using PLUSManaged to Detect Proxy Server Requirements

PLUSManaged includes a `InternetConnectionInformation` class (used by `WebServiceHelper`), which is designed to help you detect whether or not a proxy server and proxy authentication are required for your application to reach a URL. Note that this class will perform a test HTTP request using the URL (or **www.softwarekey.com** by default) as soon as it is created, which may result in some delay.

Using Specified Proxy Server Credentials

Alternatively, the .NET Framework also contains supporting classes, which you may use with web service objects (such as `XmlActivationService`) directly. Doing this might mean you may need to ask your users to configure proxy server information for your application when applicable, but could help avoid possible delays sometimes experienced during automatic testing. If your application already knows what host name (or IP address) and port to use for the proxy server, then you can set this using code similar to the following.

C#

```
XmlActivationService ws = new XmlActivationService();
ws.Proxy = new WebProxy("http://domain.com:80", true);
ws.Proxy.Credentials = new NetworkCredential("[username]", "[password]");
```

VB.NET

```
Dim ws As New XmlActivationService()
ws.Proxy = New WebProxy("http://domain.com:80", true)
ws.Proxy.Credentials = New NetworkCredential("[username]", "[password]")
```

In the example above, `"http://domain.com:80"` should be replaced with the server's host name, followed by the colon, and then the port number. Additionally, `[username]` and `[password]` should be replaced with the data entered by your users. Additionally, `WebProxy` and `NetworkCredential` both reside in the `System.Net` namespace, which you may need to add to your `using` (C#) or `Imports` (VB.NET) statements at the top of your source code.

PLUSManaged: Adding Support for Volume and Downloadable Licenses

Volume licenses are very permissive licenses which are designed to be used by anyone in a given organization without requiring each user to activate. Downloadable licenses with trigger code activation contain the same content as volume licenses, but these require **trigger code** validation to be authorized on each system. (This serves as a way to activate systems that need to be activated without any Internet connectivity, while avoiding the limitations of the very small payload size of trigger code activation.)

Implementing Support

To implement support for volume and downloadable licenses, you will need to change the way your application validates its system identifiers (especially if you want to support normal, activated licenses as well). To begin, you will need to being by overriding the `License.Load` method in your **license implementation class** so the `CurrentIdentifiers` are updated any time you load a new license file. The code excerpt below shows how you can implement support for these kind of licenses into the PLUSManaged read-onyl license sample project.

C#

```
public override bool Load(string data)
{
   bool result = base.Load(data);

   //Initialize the identifier needed to validate the volume license.
   CurrentIdentifiers.Clear();
   LicenseIDIdentifierAlgorithm algorithm = new LicenseIDIdentifierAlgorithm();
   CurrentIdentifiers.Add(algorithm.GetIdentifier(LicenseID)[0]);

   return result;
}
```

VB.NET

```
Public Overrides Function Load(data As String) As Boolean
  Dim result As Boolean = MyBase.Load(data)

  'Initialize the identifier needed to validate the volume license.
  CurrentIdentifiers.Clear()
  Dim algorithm As New LicenseIDIdentifierAlgorithm()
  CurrentIdentifiers.Add(algorithm.GetIdentifier(LicenseID)(0))

  Return result
End Function
```

After calling the base `Load` method, you could conditionally run the additional code based on the product option type by evaluating the `ProductOption.OptionType` property (which will tell you whether or not it is a volume license, or downloadable license with trigger code validation).

Additional logic is necessary for downloadable licenses with trigger code validation, as these licenses usually involve the use of two license files: one that is essentially the same as a volume license file, and a second, writable license file that is validated like a typical, activated license file (since it is technically activated using trigger codes). The **PLUSManagedGui** self-signed license sample includes support for these types of licenses, and includes sample code that can be referenced and re-used for your protected applications.

PLUSManaged: Network Floating via Semaphores

Network Floating Licensing is where an application may be restricted to running on a specific network, and restricted to a certain number of concurrent seats (where a seat may be a user or running instance of your application). This topic guides you through some basics of getting started with using network floating via semaphore files on a Windows share on a local area network (LAN). This feature has some limitations, so it is very important to review the overview of **network floating** before using it.

Getting Started

To begin, use/import the `com.softwarekey.Client.Licensing.Network` namespace to your relevant source file(s). This is typically added to your application's primary form, or where your primary application start-up logic is located.

C#

```
using com.softwarekey.Client.Licensing.Network;
```

VB.NET

```
Imports com.softwarekey.Client.Licensing.Network
```

Next, declare a private member variable in the relevant class file using the `NetworkSemaphore` class, and initialize it to `null` (C#) or `Nothing` (VB.NET).

C#

```
public class MyClass
{
   private NetworkSemaphore _Semaphore = null;
}
```

VB.NET

```
Public Class MyClass
   Private _Semaphore As NetworkSemaphore = Nothing
End Class
```

Opening a Semaphore

Opening a semaphore is synonymous with attempting to allocate a seat, which is only allowed to occur if the seats are not already consumed by other users and/or application instances. A rough example of the contents of a function that opens a semaphore is shown below.

C#

```
if (null != _Semaphore)
{
   //TODO: A semaphore is already open. You may optionally update controls and/or show a mes-
   sage here.
   return;
}

_Semaphore = new NetworkSemaphore("[\\Server\Path\To\Semaphores]", "[Prefix]", [Seats
Allowed], false, 0, false);
if (!_Semaphore.Open())
```

```
{
   //TODO: Add asynchronous semaphore searching here. See the samples for guidance.
   //TODO: The semaphore could not be opened. Report the error in _Semaphore.LastError to the
   user.
   _Semaphore = null;
   return;
}

//TODO: Update the status of controls and text shown on your form to reflect the current
state of the semaphore.
```

.NET

```
If Not _Semaphore Is Nothing Then
   'TODO: A semaphore is already open. You may optionally update controls and/or show a mes-
   sage here.
   Return
End If

_Semaphore = New NetworkSemaphore("[\\Server\Path\To\Semaphores]", "[Prefix]", [Seats
Allowed], False, 0, False)
If Not _Semaphore.Open() Then
   'TODO: Add asynchronous semaphore searching here.
   'TODO: The semaphore could not be opened. Report the error in _Semaphore.LastError to the
   user.
   _Semaphore = Nothing
   Return
End If

'TODO: Update the status of controls and text shown on your form to reflect the current state
of the semaphore.
```

Of course, there are a few blanks for you to fill in for the example above to work (or even compile):

- `"[\\Server\Path\To\Semaphores]"` needs to be replaced with a UNC path to the server and share that will host the semaphore files. Alternatively, you can specify a drive which is mapped to a Windows share. The sample applications include a dialog to let you specify this path at runtime, which you may re-use in your application if appropriate.
- `"[Prefix]"` needs to be replaced with the semaphore filename prefix. The samples default to "sema", which results in semaphore files being named like sema000.net, sema001.net, sema002.net, etc... By specifying a different prefix for each of your protected applications, different applications can use the same share location/path to manage their semaphore files (which can be convenient for your customers, as it can simplify configuration and management).
- `[Seats Allowed]` needs to be replaced with the maximum number of seats allowed (or the maximum number of semaphores that can be created). This value will usually come from your license, typically from the `License.LicenseCounter` value.

Make sure the semaphore remains open as long as the user is accessing the feature or application being licensed. Additionally, as long as your semaphore remains open, you may validate it periodically and/or any time important features are used. You can check that your application still has a valid lock on the semaphore using the `IsValid` property.

Closing a Semaphore

When your user finishes using the application or feature being licensed, you should close the semaphore to make the seat available to other users. A rough example code excerpt showing how to close a semaphore is shown below.

C#

```
if (null == _Semaphore)
{
   //TODO: The semaphore is already closed. Optionally update controls and/or show a message
   as needed.
   return;
}

_Semaphore.Close();
_Semaphore = null;

//TODO: Update the status of controls and text shown on your form to reflect the current
state of the semaphore.
```

VB.NET

```
If _Semaphore Is Nothing Then
 'TODO: The semaphore is already closed. Optionally update controls and/or show a message as
needed.
 Return
End If

_Semaphore.Close()
_Semaphore = Nothing

'TODO: Update the status of controls and text shown on your form to reflect the current state
of the semaphore.
```

Note that, even if you give the user an option to close the semaphore, you should consider automatically closing it when the relevant application or feature is being closed to help ensure seats become available to other users as appropriate.

System Identification

System identifiers are what allow you to bind a license to a particular "system." Read more about **adding basic copy protection** for an overview on how this works and is implemented. In most cases, the system being licensed is generally a single device. However, in the case of network floating licensing, the "system" that is authorized to use the license is typically a network of devices. Naturally, this difference means system identification is usually handled differently when using network floating licensing via semaphores.

Binding the license to the share

To begin, PLUSManaged includes a `NetworkNameIdentifierAlgorithm` class, which is designed to generate an identifier using a given server name and share name in a UNC path (e.g. \\SERVER1\\SHARE1). This algorithm can also accept a mapped network drive configured to use a UNC path and to reconnect at login.

Important

While NetworkNameIdentifierAlgorithm is import to use to prevent casual copying of licenses from one file share to another, this by itself is not enough to make things completely tamper-proof. This is because of how it is technically possible to create another server with the same name and share name on a different segment or network, and how it is possible to copy sub-directories on a share. Furthermore, these concerns are exacerbated by the ubiquity of virtualization technologies. We strongly recommend using NetworkNameIdentifierAlgorithm with additional system identifier algorithms to strengthen copy protection in your applications. We also recommend you consider using **Cloud Controlled Network Floating Licensing** instead, if possible.

Binding the license to the server's volume

A second system identifier algorithm PLUSManaged includes is HardDiskVolumeSerialIdentifierAlgorithm, which is designed to get the serial number (typically determined when a drive is formatted) of a volume/drive. While this is most often used for activating individual devices, it is also possible to use this to generate an identifier for a remote drive's volume serial. Below is some example code showing how you can implement this in PLUSManaged:

C#

```
_License.CurrentIdentifiers.Add(HardDiskVolumeSerialIdentifier.GetIdentifier("[driveLetter]:\"));
```

VB.NET

```
_License.CurrentIdentifiers.Add(HardDiskVolumeSerialIdentifier.GetIdentifier("[driveLetter]:\"))
```

Even though this algorithm further strengthens your application's copy protection, it too is not completely tamper-proof. The volume serial number is typically determined when a drive is formatted. It is possible to duplicate this when using a system imaging tool (such as Norton Ghost) to restore a drive image backup to a new drive, copying a virtual machine to another host, or just using a third-party tool to alter a volume's serial. While these approaches do require some technical expertise, it is not outside of the capabilities of the typical IT staff who would be responsible for installing and configuring your application for a network. Consequently, we again strongly advise you to consider using additional algorithms, such as binding to application-specific data, to further strengthen copy protection for your applications.

Binding to application-specific data

In many cases, it is possible to bind a license to some information that is unique to the customer to which the license was issued. For example, if your software already has to ask the user for some sort of business license number, your software could **implement a custom identifier** that leverages that information. While this does not necessarily prevent abuse from within a given organization, it is a good way to prevent abuse from spreading outside the organization.

Binding to an absolute path

As noted in the "Binding the license to the share", binding the license to a server name and share is not enough to prevent abuse. Sometimes it is necessary to take additional measures to prevent abuse of a license from within an organization, which is not always practical to achieve by binding to application-specific data (as noted in the section immediately above). To prevent abuse on an organizations server, you can do the following:

- Ask the organization to provide the semaphore path ahead of time. Be careful to request an absolute UNC path and not a path that uses a mapped drive letter here, as this will be more reliable.
- Enter the absolute path given in the step above into either the license's custom data field or a user defined field in SOLO Server.

- Only provide the users with a License ID and password for activation (or with a license file if using volume licensing) after the above 2 steps have been completed.
- Your application's license validation logic would then be configured to require use of this path as the semaphore and license file path. If the path being used does not appear to be what is expected, your application would then display an error message that makes it clear to you (and your customer service staff) that this is why your software's license validation failed.

Review the Samples

The **sample applications** include much more functionality, including:

- Asynchronous searching for available semaphores; which can be important for high-load environments, and environments where the number of seats used often nears or reaches the maximum allowed.
- Background cleanup thread support, which helps minimize the presence of any orphaned semaphore files that may have been left behind from a computer locking up, or the application terminating abnormally.
- Complete example integration code, showing you how you can tie your network floating logic to a license, and use properties that come down from SOLO Server to make it very quick and easy to adjust things like the number of seats allowed through license refreshing (see the **ELM** overview for details).

PLUSManaged: Cloud-Controlled Floating Network Licensing using SOLO Server

Network Floating Licensing is where an application may be restricted to running on a specific network, and restricted to a certain number of concurrent seats (where a seat may be a user or running instance of your application). This topic guides you through some basics of getting started with using Cloud-Controlled Floating Network Licensing using SOLO Server. This feature has some limitations, so it is very important to review the overview of **network floating** before using it.

Important

Note that the Cloud-Controlled Floating Network Licensing using SOLO Server is only supported in Windows platforms. See the overview of **network floating** for additional details.

If you wish to use Cloud-Controlled Floating Network Licensing using SOLO Server, you will need to **contact us** for additional details on availability and pricing.

Getting Started

To begin, use/import the necessary namespaces to your relevant source file(s). This is typically added to your application's primary form, or where your primary application start-up logic is located.

C#

```
using com.softwarekey.Client.Licensing;
using com.softwarekey.Client.Licensing.Network;
using com.softwarekey.Client.Utils;
using com.softwarekey.Client.WebService.XmlNetworkFloatingService;
```

VB.NET

```
Imports com.softwarekey.Client.Licensing
Imports com.softwarekey.Client.Licensing.Network
Imports com.softwarekey.Client.Utils
Imports com.softwarekey.Client.WebService.XmlNetworkFloatingService
```

Next, you'll need to add a member variable for the network session, and some static configuration properties.

C#

```
private NetworkSession _Session = null;
//TODO: Adjust the certificate path as needed.
private static string _CertificatePath = Path.Combine(Path.GetDirectoryName(Application.Ex-
ecutablePath), "Session.xml");

//TODO: Adjust this to use your SOLO Server account's encryption key data.
private static AuthorEncryptionKey EncryptionKey
{
   get { return new AuthorEncryptionKey("[Your Envelope Key from SOLO Server", "Your Envelope
   from SOLO Server", false); }
}

//TODO: Adjust the algorithms selected to suit your needs.
private static List<SystemIdentifierAlgorithm> SystemIdentifierAlgorithms
{
```

```
    get
    {
        return new List<SystemIdentifierAlgorithm>(new SystemIdentifierAlgorithm[] {
            new HardDiskVolumeSerialIdentifierAlgorithm(HardDiskVolumeSerialFilterType.Oper-
            atingSystemRootVolume),
            new NicIdentifierAlgorithm(),
            new ComputerNameIdentifierAlgorithm() });
    }
}
```

VB.NET

```
Private _Session As NetworkSession = Nothing
'TODO: Adjust the certificate path as needed.
Private Shared _CertificatePath As String = Path.Combine(Path.GetDirectoryName(Application.Ex-
ecutablePath), "Session.xml")

'TODO: Adjust this to use your SOLO Server account's encryption key data.
Private Shared ReadOnly Property EncryptionKey() As AuthorEncryptionKey
    Get
        Return New AuthorEncryptionKey("[Your Envelope Key from SOLO Server", "Your Envelope
        from SOLO Server", False)
    End Get
End Property

'TODO: Adjust the algorithms selected to suit your needs.
Private Shared ReadOnly Property SystemIdentifierAlgorithms() As List(Of Sys-
temIdentifierAlgorithm)
    Get
        Return New List(Of SystemIdentifierAlgorithm)(New SystemIdentifierAlgorithm() {New
        NicIdentifierAlgorithm(), _
            New HardDiskVolumeSerialIdentifierAlgorithm(HardDiskVolumeSerialFilterType.Oper-
            atingSystemRootVolume), _
            New ComputerNameIdentifierAlgorithm()})
    End Get
End Property
```

Note that the samples include encryption envelopes that are linked to the Instant SOLO Server **test account**. You can either use the test credentials, or you can retrieve the encryption key data **for your account**, even if you are currently evaluating Instant SOLO Server.

Opening a Session

Opening a session is synonymous with attempting to allocate a seat, which is only allowed to occur if the seats are not already consumed by other users and/or application instances. A rough example of the contents of a function that opens a session is shown below.

C#

```
_Session = new NetworkSession(EncryptionKey, true, true, SystemIdentifierAlgorithms, _Cer-
tificatePath, false);
if (_Session.OpenSession([LicenseID], "[Customer Password]"))
{
```

```
   //TODO: Session opened successfully - add additional logic here to update your dialog's
   controls.
}
else
{
   //TODO: Handle error condition. Show an error message to the user with details from _Ses-
   sion.LastError.
}
```

VB.NET

```
_Session = new NetworkSession(EncryptionKey, True, True, SystemIdentifierAlgorithms, _Cer-
tificatePath, False)
If _Session.OpenSession([LicenseID], "[Customer Password]") Then
   'TODO: Session opened successfully - add additional logic here to update your dialog's con-
   trols.
Else
   'TODO: Handle error condition. Show an error message to the user with details from _Ses-
   sion.LastError.
End If
```

Upon opening a session, you receive a network session certificate, which contains data about the session and how the protected application should behave. For example, it contains the date and time the session expires, whether or not it has been checked-out, how long it should wait between poll attempts, and more. In the example above, the data is available in properties under the `_Session.Certificate` property.

Validating a Session

Once a session is open, and immediately after any action (other than closing the session) is performed, it is necessary to validate the session. The example below shows how you can use the `NetworkSessionValidation` class, which is included with PLUSManaged to simplify the validation process.

C#

```
NetworkSessionValidation validation = new NetworkSessionValidation(_Session);
if (validation.Validate())
{
   //TODO: Your session is valid. Update controls as appropriate.
}
else
{
   //TODO: Notify the user that the session is no longer valid, update controls as appro-
   priate, and include details from validation.LastError.
   //TODO: Close the session if it is already open.
}
```

VB.NET

```
Dim validation As New NetworkSessionValidation(_Session)
If validation.Validate() Then
   'TODO: Your session is valid. Update controls as appropriate.
Else
   'TODO: Notify the user that the session is no longer valid, update controls as appro-
   priate, and include details from validation.LastError.
   'TODO: Close the session if it is already open.
End If
```

Note that it is important to close the session rather than abandoning it when validation fails. This is because the session should not be allowed to be used when invalid, and if the session remains open on SOLO Server even though local validation has failed, abandoning it will cause SOLO Server to consider the seat to be valid until the allocated until date (or the date and time the session is set to expire if it has not polled successfully before then).

Polling a Session

When a session is open, it is set to expire by its "allocated until date." In other words, if no poll occurs before the date and time shown in the "allocated until date" is reached, the session is considered expired by SOLO Server. Successfully polling the session with SOLO Server automatically extends the allocated until date, which is what allows SOLO Server to recover orphaned sessions/seats (which is helpful for handling abnormal application termination).

C#

```
if (_Session.Poll())
{
   //TODO: The poll was successful.  Re-validate the session and update controls as needed.
}
else
{
   //TODO: The poll failed.  Show an error message and close the session.
}
```

VB.NET

```
If _Session.Poll() Then
   'TODO: The poll was successful.  Re-validate the session and update controls as needed.
Else
   'TODO: The poll failed.  Show an error message and close the session.
End If
```

If a session has been checked-out for extended/offline use, you may still poll to verify the session if Internet connectivity is available. However, the "allocated until date" is not automatically extended for a checked-out session.

Checking-out and Checking-in a Session

In some cases, you may find the need to allow users to check-out a session/seat for offline and/or extended use. An example could be a user who needs to use the protected application while traveling. To allow this, you can allow the user to check-out a session for a requested duration. When successful, SOLO Server will respond with an updated session certificate that has an allocated until date that is valid for the requested duration (or until the minimum or maximum check-out duration allowed if the requested duration is outside of those bounds).

C#

```
if (!_Session.Certificate.CheckedOut)
{
   //TODO: The session is already checked-out; optionally show an error message.
   return;
}

if (_Session.CheckoutSession([requestedCheckoutDuration]))
{
   //TODO: The session was checked-out successfully.  Re-validate the session and update con-
   trols as needed.
}
else
{
   //TODO: The session could not be checked-out. Show an error message including details from
   _Session.LastError.
}
```

VB.NET

```
If _Session.Certificate.CheckedOut Then
   'TODO: The session is already checked-out; optionally show an error message.
   Return
End If

If _Session.CheckoutSession([requestedCheckoutDuration]) Then
   'TODO: The session was checked-out successfully.  Re-validate the session and update con-
   trols as needed.
Else
   'TODO: The check-out failed.  Show an error message including details from _Ses-
   sion.LastError.
End If
```

Once checked-out, your application may continue to poll when Internet connectivity is available (just keep in mind that attempting to poll while offline could cause performance degradation). The seat will then be consumed until the allocated until date is reached, or the user checks-in and closes the session. So in the example of a traveling user, you might allow the user to check-out for up to a week, even though the user is only expected to travel for 3 to 4 days. This allows the user to continue to use the application in a disconnected state should unforeseen circumstances (such as inclement weather) arise. If the user returns when expected, he or she could then check-in the session and return to normal connected use so the seat becomes available for other users when not in use.

C#

```
if (!_Session.Certificate.CheckedOut)
{
   //TODO: The session is not checked-out; optionally show an error message.
   return;
}

if (_Session.CheckinSession())
{
   //TODO: The session was checked-in successfully.  Re-validate the session, and update con-
   trols as needed.
}
else
```

```
{
   //TODO: The session could not be checked-in. Show an error message including details from
   _Session.LastError.
}
```

VB.NET

```
If Not _Session.Certificate.CheckedOut Then
   'TODO: The session is not checked-out; optionally show an error message.
   Return
End If

If _Session.CheckinSession() Then
   'TODO: The session was checked-in successfully.  Re-validate the session, and update con-
   trols as needed.
Else
   'TODO: The session could not be checked-in.  Show an error message including details from
   _Session.LastError.
End If
```

Once the session has been checked-in, it may resume normal connected use with periodic polls. Otherwise, the session should be closed if the session is no longer being used actively.

Closing a Session

A session should be closed any time it is no longer being used by the user, or any time it fails local validation. This helps ensure it the seat that was previously in use becomes free for other users to open and use.

C#

```
if (_Session.Close())
{
   //TODO: The session was closed successfully.  Update controls as needed.
}
else
{
   //TODO: The session could not be closed. Show an error message including details from _Ses-
   sion.LastError, and close the session.
}
```

VB.NET

```
If _Session.Close() Then
   'TODO: The session was closed successfully.  Update controls as needed.
Else
   'TODO: The session could not be closed. Show an error message including details from _Ses-
   sion.LastError, and close the session.
End If
```

Review the Sample

The example code excerpts given above are only meant to provide guidance on how the APIs should be used. Review the **PLUSManaged samples** to see complete, functioning example code that can be re-used in your applications.

PLUSManaged: Deploying Protected Applications

Review the **system requirements** for details on what is required to run PLUSManaged. If your application also uses PLUSManagedGui, be sure to read about **deploying applications that use PLUSManagedGui**.

Distributing the PLUSManaged Library

The PLUSManaged should be installed in the same directory as your application (or .exe file). This greatly simplifies deployment and also prevents your application from using other versions the library that may not be compatible with your application (which could be installed by other applications that use PLUSManaged).

Important

Although installing the library to the application directory is **strongly** recommended, it is possible to install the assembly to the **Global Assembly Cache** (GAC) if required by your application.

If your application requires you to install dependencies in the GAC, make sure your application's reference to any Protection PLUS 5 SDK libraries (such as PLUSManaged or PLUSManagedGui) has the *Specified Version* property set to true.

Serialization Assemblies

PLUSManaged does include an XML serialization assembly, named *PLUSManaged.XmlSerializers.dll*. Although you are not required to ship this assembly with your application, doing so is strongly recommended, as it can significantly increase your application's performance during start-up. If this assembly is not distributed with your application, then .NET will automatically compile a serialization assembly each time your application starts-up (which can cause some delay).

Obfuscation

Important

The Protection PLUS 5 SDK .NET libraries (such as PLUSManaged and PLUSManagedGui) are obfuscated, but this alone cannot prevent a hacker from modifying your application's source code. When your .NET application is compiled, it is compiled to what is called Microsoft Intermediate Language (MSIL - pronounced like "missile") code. There are tools that can be used to translate and reverse engineer MSIL code, so it is imperative that you select and use an obfuscation tool to further protect your application and your investments in it. Microsoft offers a **good summary** of this subject.

Assembly Embedding

It is possible to obfuscate your application in a way which embeds the Protection PLUS 5 SDK libraries with your application. This is recommended where possible, as it works with PLUSManaged and PLUSManagedGui, provides some extra security with most obfuscation tools, and can help simplify your deployment (as won't be necessary to distribute the library once it is embedded in your application).

Installation Requirements

Protection PLUS 5 SDK license files are generally not distributed with an application. When using read-only license files, they must be issued by SOLO Server during activation. When using writable license files, such as for unmanaged evaluations/trials that do not require activation, the license files are generated the first time the application runs. Depending on where you choose to store your application's license files, you may need to set permissions on these locations during installation so that your application has write access and can save it's license files when it runs

or is activated. Permissions can be set on the directory or file level. If you desire to set permissions on the file level, you can install a blank (empty) license file as a placeholder during installation.

Writable Licenses

When using writable licenses, PLUSManaged automatically attempts to set permissions on the license file and the aliases when they are saved. You can use this to your advantage by creating a separate helper application creates and saves the initial the license file and aliases, or add a custom command-line switch to your application that allows it to do this silently. This would enable you to call the helper application or your main application (with the command-line switch) from your installer.

Important

Microsoft Office applications (such as Word, Excel, Access, etc...) may run in a **ClickToRun** environment. This environment has known limitations that make it problematic for licensed Office add-ins and macros to use global locations. Consequently, licensed add-ins and macros that target these environments should only use user-specific locations for licenses and aliases. See **this knowledge-base article** for more details.

Using .NET to Set Permissions in Windows

PLUSManaged includes a `IOHelper.GrantControlToWorld` method that can be used to grant everyone permissions to a file. However, you may wish to write your own application that initializes permissions for other things as well. The .NET Framework offers a number of classes which may be used to set permissions on files, folders, and registry keys. You can use these objects to create a separate helper application, which may be called by your installer. The easiest way to do this is to give "Everyone" access to read and write to the license file and aliases. To do this you can begin by calling the `IOHelper.GetEveryoneAccountNameString` method in PLUSManaged, or by using code similar to the example below to get the appropriate account name (which can vary depending on the computer's regional settings). Note that, to use the example code provided here, you may need to add a reference to the System.Security .NET assembly to your project, and add a using or Imports statement for the `System.Security.AccessControl`, `System.Security.Principal`, and `Microsoft.Win32` namespaces at the top of your source file.

C#

```
SecurityIdentifier sid = new SecurityIdentifier(WellKnownSidType.WorldSid, null);
NTAccount acct = sid.Translate(typeof(NTAccount)) as NTAccount;
string everyoneAccountName = acct.ToString();
```

VB.NET

```
Dim sid As New SecurityIdentifier(WellKnownSidType.WorldSid, Nothing)
Dim acct As NTAccount = CType(sid.Translate(Type.GetType("Sys-
tem.Security.Principal.NTAccount")), NTAccount)
Dim everyoneAccountName As String = acct.ToString()
```

Once you have the appropriate account name (in `everyoneAccountName`), you can use it to set permissions on a file or registry key as shown in the examples below.

Setting Permissions on a File

C#

```
FileSecurity sec = File.GetAccessControl("[Path to File]");
sec.AddAccessRule(new FileSystemAccessRule(everyoneAccountName, FileSystemRights.FullControl,
```

```
AccessControlType.Allow));
File.SetAccessControl("[Path to File]", sec);
```

VB.NET

```
Dim sec As FileSecurity = File.GetAccessControl("[Path to File]")
sec.AddAccessRule(new FileSystemAccessRule(everyoneAccountName, FileSystemRights.FullControl,
AccessControlType.Allow))
File.SetAccessControl("[Path to File]", sec)
```

Setting Permissions on a Registry Key

C#

```
RegistryKey key = Registry.LocalMachine.OpenSubKey("[Path to Registry Key]", true);
RegistrySecurity sec = new RegistrySecurity();
sec.GetAccessRules(true, true, typeof(System.Security.Principal.SecurityIdentifier));
sec.AddAccessRule(new RegistryAccessRule(everyoneAccountName, RegistryRights.FullControl,
AccessControlType.Allow));
key.SetAccessControl(sec);
key.Close();
```

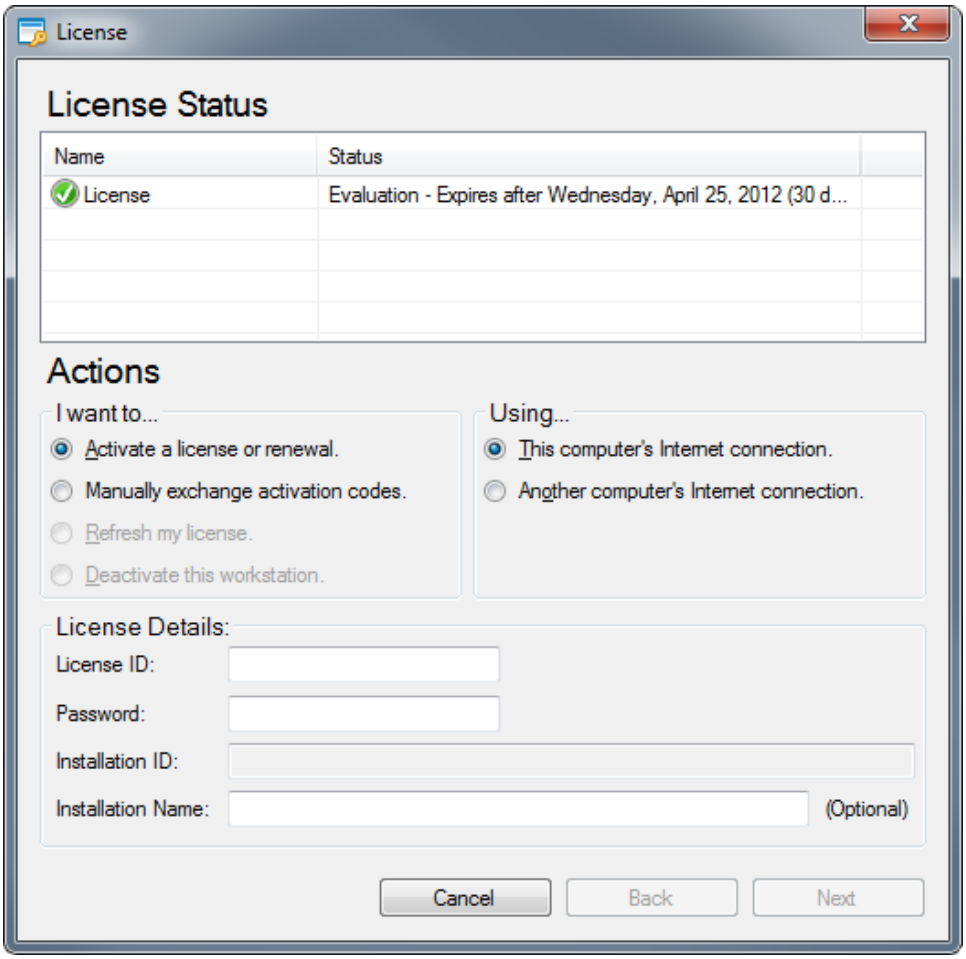
VB.NET

```
Dim key As RegistryKey = Registry.LocalMachine.OpenSubKey("[Path to Registry Key]", True)
Dim sec As New RegistrySecurity()
sec.GetAccessRules(True, True, Type.GetType("System.Security.Principal.SecurityIdentifier"))
sec.AddAccessRule(New RegistryAccessRule(everyoneAccountName, RegistryRights.FullControl,
AccessControlType.Allow))
key.SetAccessControl(sec)
key.Close()
```

PLUSManagedGui Overview

Creating user interfaces to allow users to manage your application licenses involves design, development, and testing. PLUSManagedGui is a fully managed, .NET component that provides you with an easy-to-use, pre-built GUI that you can simply plug right into your applications. PLUSManagedGui is designed for use with applications that use PLUSManaged and are built on or able to use `System.Windows.Forms` GUIs (PLUSManagedGui is not designed to be integrated into services or web applications directly). This applies to Protection PLUS 5 SDK .NET Edition.

Below is a screen shot of the kind of user interface PLUSManagedGui adds to your application:



Important

Throughout this section of the manual, the `SampleLicense` class is referenced. This class and its supporting classes are re-used from the **sample applications**.

This section of the manual focuses on showing you how to use PLUSManagedGui, and does not focus on showing you **how to use PLUSManaged** directly. A complete reference for PLUSManagedGui is also available in the **PLUSManaged API reference**.

Summary of Features

PLUSManagedGui is a visual component that is very easy to use. After **adding a reference** to your project, and simply dragging-and-dropping the component onto your application's primary dialog/form, you can configure properties and settings visually using the properties pane in Visual Studio. Some of the many, time-saving features available in PLUSManagedGui are summarized below.

Simplified Processing

PLUSManagedGui allows your users to select which actions to take and specify how to go about performing them (both with and without direct Internet connectivity) through a common *license management dialog*. When a user requests an action, PLUSManagedGui automatically performs it for you and passes the results to you in a single, simplified event. This same functionality also provides the flexibility you need to make decisions about the final result of the action performed, and simplifies reporting the final result to your end-users.

SOLO Server Integration

PLUSManagedGui calls the SOLO Server web services for you automatically based on which action the user decides to take through the built-in GUI.

Proxy Server Support

Support for proxy servers and proxy authentication can be particularly important when supporting larger clients, especially organizations like hospitals, academic institutions, government or military organizations, etc... PLUSManagedGui automatically supports using the same proxy server configuration used by Internet Explorer on Windows systems, and also automatically handles obtaining and using users' proxy authentication credentials. The gathered information may even be used for other, in-house or third-party web services you may call in your application.

Support for Disconnected Systems

Supporting activation and automating license management of systems which have no or very limited Internet connectivity is greatly simplified.

Manual Request Files

When a user needs to activate, refresh, or deactivate his or her license, but lacks an Internet connection on the licensed system, he or she may generate a manual request file. A manual request file is simply an HTML file that the user may copy to another system or device (typically using removable media such as a USB thumb-drive or a portable hard drive) that has an Internet connection. Once copied, the user may double-click the HTML file and save the result, and load the result in the computer that is being licensed. The system which is being activated or licensed is the only one that needs to have your software installed. The only requires for using the manual request file on other systems or devices is a web browser and a working Internet connection.

Trigger Codes

If you need to support environments that completely lack Internet connectivity, you may need a means of activating your application's licenses by phone or fax. Trigger Codes are the solution for this scenario, as they allow you to activate customers quickly and easily using activation codes that can easily be read by and to your customers verbally.

Customizable Splash Screen

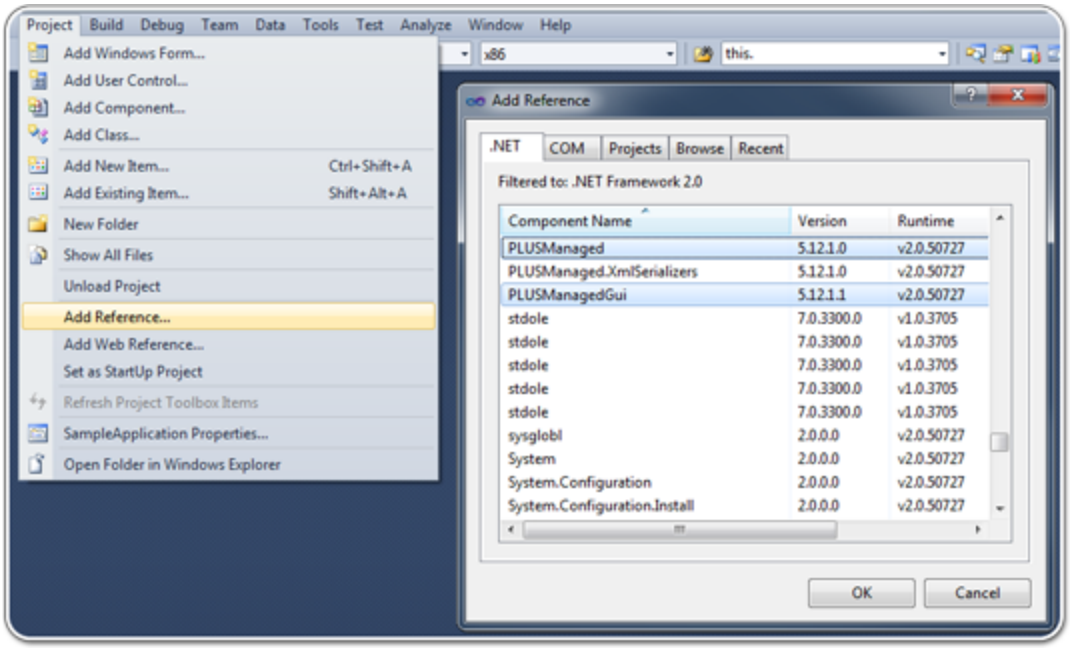
Many (if not most) applications check the status of the license and/or refresh the license file with SOLO Server regularly. It is often important that this validation occurs while your application is starting, but this can often take some

time (especially if the customer's Internet connection is slow). Additionally, there may be other start-up logic in your application that could take a while to complete. **Displaying a splash screen** while your application performs these tasks is not only convenient, it is considered a best practice. The splash screen lets users know your application has been started and is responsive while the application performs its start-up tasks. PLUSManagedGui includes an easy-to-use splash screen that can be used to load and validate your licenses, as well as perform any additional start-up logic required by your application while display a customized graphic of your choice (PNG images using alpha-transparency are supported).

Adding PLUSManagedGui to your application

Adding References

Before you can use PLUSManagedGui with your application, you need to add references for PLUSManaged and PLUSManagedGui to your project. Begin by clicking *Project/Add Reference* from the menu in Visual Studio (as shown below). Then, click the .NET tab, select PLUSManaged and PLUSManagedGui from the list (you can hold the *Ctrl* key on your keyboard while clicking to select them both), and click OK.

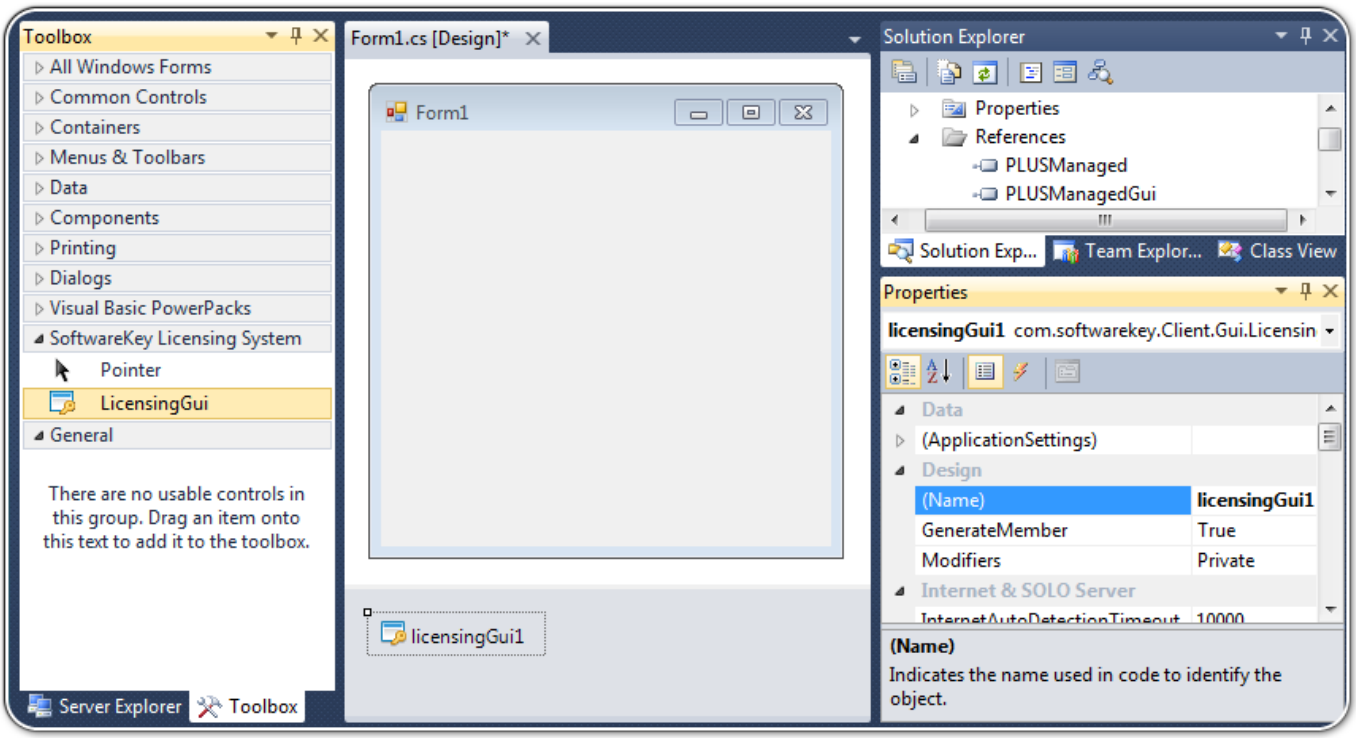


Adding the Component

Once you have added the required references to your project, the next step is to add an instance of the PLUSManagedGui component to your application.

Using the Designer

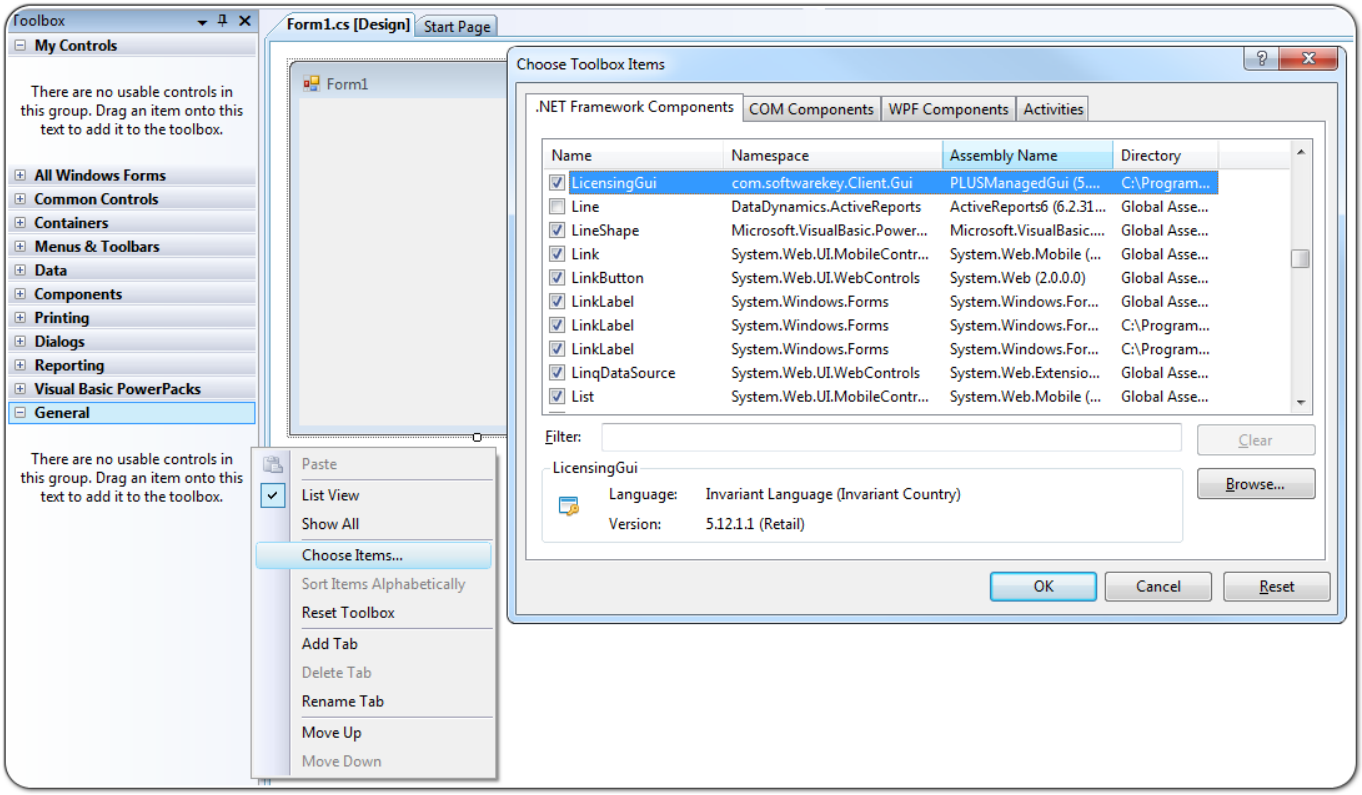
The easiest way to add the PLUSManagedGui component to your applications is to use Visual Studio's designer. To do this, double-click on your application's primary/main form in Solution Explorer to open it in the designer. Then, open/expand the Toolbox, find the SoftwareKey Licensing System category, and double-click on the `LicensingGui` control to add it to the form as shown below.



From here, you will be able to click on the component and configure its properties using the designer (on the right, just under the Solution Explorer in the above screen shot).

Older Versions of Visual Studio

If you are earlier using Visual Studio, such as 2005 (8.0) or 2008 (9.0), then the icon will not be added to your Toolbox automatically. To add it to your Toolbox, right click in it, click *Choose Items...*, select the *.NET Framework Assemblies* tab, and select the LicensingGui component from the list as shown below.



If you do see the component on the list, you can click the Browse button and navigate to the Protection PLUS 5 SDK installation folder to find and select PLUSManagedGui.dll.

Using Source Code

Although using the designer is the easiest way to add the component to your application, it is possible to use the component from source code alone (as the designer simply generates this code for you automatically). An example of what this could look like in a basic form's source code is provided below.

VB.NET

```
Imports com.softwarekey.Client.Gui

Public Class Form1
   Private licensingGui1 As New LicensingGui()
End Class
```

C#

```
using System.Windows.Forms;
using com.softwarekey.Client.Gui;

namespace SampleApplication
{
   public partial class MainForm : Form
   {
      private LicensingGui licensingGui1 = new LicensingGui();

      public MainForm()
      {
         InitializeComponent();
      }
   }
}
```

PLUSManagedGui: Initializing the Component

After adding PLUSManagedGui to your application, the next step is to initialize the component. If the component is not initialized correctly, any attempt to use PLUSManagedGui while your application is running will result in an error message being displayed.

Important

Throughout this section of the manual, the `SampleLicense` class is referenced. This class and its supporting classes are re-used from the **sample applications**.

C#

```
private bool m_LastLicenseValidationResult = false;
private SampleLicense m_License;

public MainForm()
{
   InitializeComponent();
   m_License = new SampleLicense(licensingGui1);
   licensingGui1.ApplicationLicense = (License)m_License;
}
```

VB.NET

```
Private m_LastLicenseValidationResult As Boolean = False
Private m_License As SampleLicense

Public Sub New()
   InitializeComponent()
   m_License = New SampleLicense(licensingGui1)
   licensingGui1.ApplicationLicense = m_License
End Sub
```

In the example above, the `SampleLicense` class constructor receives and stores the reference to the PLUSManagedGui component. This allows your license implementation logic to leverage some of the features in PLUSManagedGui. For example, this will allow your license implementation class to automatically obtain proxy authentication credentials from a user when required for web service calls.

The next noteworthy part of the above code example is that the `ApplicationLicense` property is initialized and points to your license implementation object. This step is required by PLUSManagedGui, and an error will be displayed if it is omitted.

License File Path

In order to access your license file, you will need to make your code aware of the location of the license file. The example code below shows how you can add a property to your form which assumes the license file is located in the same directory as your application (or your .EXE file).

C#

```
internal static string LicenseFilePath
{
```

```
   get { return Path.Combine(Path.GetDirectoryName(Application.ExecutablePath),
   "LicenseFile.xml"); }
}
```

VB.NET

```
Friend Shared ReadOnly Property LicenseFilePath() As String
   Get
      Return Path.Combine(Path.GetDirectoryName(Application.ExecutablePath),
      "LicenseFile.xml")
   End Get
End Property
```

Of course, when you select a final path for your application's license files, you will need to be considerate of permissions during **deployment** or installation of your application.

Initializing the License

Once the objects have been initialized per the instructions above, the next step is to initialize the license by loading and validating it as shown below.

C#

```
m_LastLicenseValidationResult = m_License.LoadFile(LicenseFilePath);
if (m_LastLicenseValidationResult)
   m_LastLicenseValidationResult = m_License.Validate();
```

VB.NET

```
m_LastLicenseValidationResult = m_License.LoadFile(LicenseFilePath)
If (m_LastLicenseValidationResult) Then
   m_LastLicenseValidationResult = m_License.Validate()
End If
```

This code can be called as the last step in your form's constructor, or it may be called while your **splash screen** is loading (if your application is configured to periodically communicate with SOLO Server).

PLUSManagedGui: Using a Splash Screen

The PLUSManagedGui component includes support for displaying a customizable splash screen image (PNG images using alpha-transparency are supported). Splash screens are fairly ubiquitous, and their purpose is to inform the user that the application is initializing. Using a splash screen is especially useful when applications perform initialization that takes some time before showing the user interface.

Update the Form Constructor

Since you already implemented some additional code in your constructor to **initialize objects** as required to use PLUSManagedGui, your constructor will need to be updated to first hide the form to prevent it from being displayed prematurely, and it will also need to show the splash screen after the objects are initialized. An example of this modified constructor code is below.

C#

```
public MainForm()
{
   //Hide the main form initially (so the user only sees the splash screen).
   Hide();

   //Initialize the form's components (this is required for your form to function properly).
   InitializeComponent();

   //Initialize the License and LicenseGui objects.
   m_License = new SampleLicense(licensingGui1);
   licensingGui1.ApplicationLicense = (License)m_License;

   //Display the splash screen.
   licensingGui1.ShowDialog(LicensingGuiDialog.SplashScreen);
}
```

VB.NET

```
Public Sub New()
   //Hide the main form initially (so the user only sees the splash screen).
   Hide()

   'Initialize the License and LicenseGui objects.
   InitializeComponent()

   'Initialize the License and LicenseGui objects.
   m_License = New SampleLicense(licensingGui1)
   licensingGui1.ApplicationLicense = m_License

   'Display the splash screen.
   licensingGui1.ShowDialog(LicensingGuiDialog.SplashScreen)
End Sub
```

Implement Initialization Logic

To initialize the application's licensing objects, the application will need to:

1. Load the license file.
2. Validate the license file (and aliases, if needed).
3. Store the validation result to minimize the number of times the license is validated.

This code would resemble the following:

C#

```
private void InitializeApplicationSettings(object sender, EventArgs e)
{
   m_LastLicenseValidationResult = m_License.LoadFile(Path.Combine(Path.GetDirectoryName
   (Application.ExecutablePath), "LicenseFile.xml"));
   if (m_LastLicenseValidationResult)
      m_LastLicenseValidationResult = m_License.Validate();

   //TODO: Add your application's initialization logic here.
}
```

VB.NET

```
Private Sub InitializeApplicationSettings(sender> As Object, e As EventArgs)
   m_LastLicenseValidationResult = m_License.LoadFile(Path.Combine(Path.GetDirectoryName
   (Application.ExecutablePath), "LicenseFile.xml"))
   If m_LastLicenseValidationResult Then
      m_LastLicenseValidationResult = m_License.Validate()
   End If

   'TODO: Add your application's initialization logic here.
End Sub
```

Please keep in mind that the example logic here is simplified, and lacks support for advanced licensing features such as support for downloadable and volume license files.

Implement Completion Logic

Any logic that should be run when the initialization logic has completed should be implemented in its own method. In many (if not most) cases, this may only include showing the application's primary form or dialog. Here is an example of what this code can look like.

C#

```
private void SplashWorkCompleted(object sender, EventArgs e)
{
   //Show the main form.
   Show();
}
```

VB.NET

```
Private Sub SplashWorkCompleted(sender As Object, e As EventArgs)
   'Show the main form.
   Show()
End Sub
```

Next, open the form in the designer, click on the form (but not any of the controls on the form), click the lightning bolt icon in the properties pane, and double-click on the Shown event to create a handler. Then, add the code similar to the following in the event handler.

C#

```
private void MainForm_Shown(object sender, EventArgs e)
{
   //TODO: Here is where you need to update controls on your form to reflect the updated
   license status as needed.

   //Check the license validation result.
   if (!m_LastLicenseValidationResult)
   {
      //If license validation failed, show the license management form when the application
      starts.
      licensingGui1.ShowDialog(LicensingGuiDialog.LicenseManagement);
   }
}
```

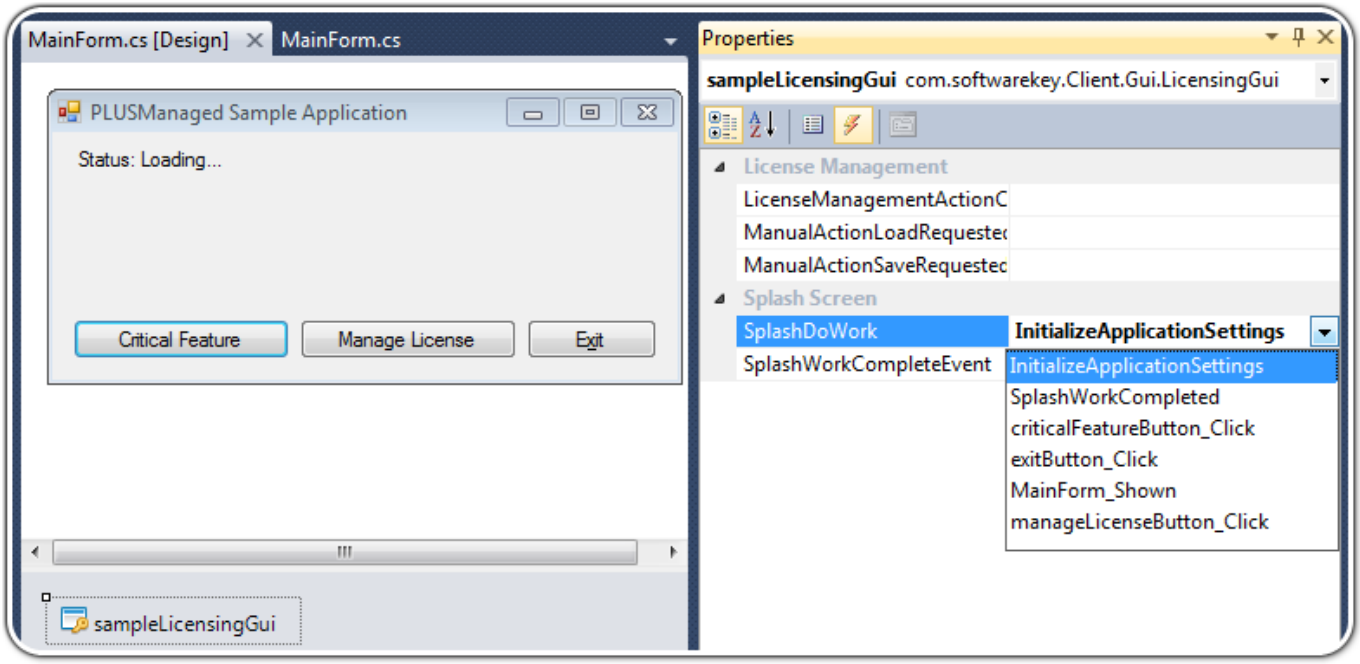
VB.NET

```
Private SubMainForm_Shown(sender As Object, e As EventArgs) Handles MyBase.Shown
   'TODO: Here is where you need to update controls on your form to reflect the updated
   license status as needed.

   'Check the license validation result.
   If Not m_LastLicenseValidationResult Then
      'If license validation failed, show the license management form when the application
      starts.
      licensingGui1.ShowDialog(LicensingGuiDialog.LicenseManagement)
   End If
End Sub
```

Wire Event Handlers

Once your initialization logic is implemented, you will need to proceed to wire method to handle an event. The easiest way to do this is to click on the `LicensingGui` component in the designer, click the lightning bolt icon in the properties pane, and select the newly created `InitializeApplicationSettings` method from the drop-down list for the `SplashDoWork` event. Also select the `SplashWorkComplete` method for the `SplashWorkCompleteEvent`.



Additional Configuration

The PLUSManagedGui component's `LicensingGui` object has several properties you can configure for customizing the look and feel of the splash screen. These properties are outlined in the table below, and are all configurable by selecting the `LicensingGui` component on your form's designer.

Property Name	Description
`SplashBackgroundColor`	The background color of the splash screen. This is the color displayed when no splash screen image is present.
`SplashImage`	If specified, this is the image that is displayed on the splash screen. The splash screen will automatically size itself to match the size of the image. PNG images using alpha-transparency are supported.
`SplashMinimumDuration`	The minimum duration of time (in milliseconds) in which the splash screen will be displayed. If your initialization logic completes sooner, the splash screen will continue to be displayed until this amount of time passes.
`SplashTopMost`	When set to true, the splash screen will be displayed on top of all other windows/dialogs..

PLUSManagedGui: Displaying the License Status

Determining the Status

The status of the license can simply be determined by the result of the license validation preformed. In the **sample applications**, the `m_LastLicenseValidationResult` member variable is used to keep track of the last validation result in the application's main form. If the validation succeeded, then the license should be considered valid. Otherwise, if the validation did not succeed, the license's `LastError` property should be evaluated for details on why the validation failed.

Customizing Error Descriptions

The `LicenseError` object returned by the `LastError` property includes an `ErrorString` property and `ToString` method that return a readable description of the error that last occurred. However, there may be cases where you want to customize the error description in your application. The code excerpt below is a small example that shows how you can customize certain error messages for your application.

C#

```
internal static string GenerateLicenseErrorString(License lic)
{
   StringBuilder status = new StringBuilder();

   switch (lic.LastError.ErrorNumber)
   {
      case LicenseError.ERROR_COULD_NOT_LOAD_LICENSE:
         status.Append(lic.LastError.ErrorNumber);
         status.Append(": License not found - activation is required.");
         break;
      case LicenseError.ERROR_WEBSERVICE_RETURNED_FAILURE:
         //Web service error message.
         status.Append(lic.LastError.ExtendedErrorNumber);
         status.Append(": ");
         status.Append(LicenseError.GetWebServiceErrorMessage(lic.LastEr-
         ror.ExtendedErrorNumber));
         break;
      default:
         //Show a standard error message.
         status.Append(lic.LastError.ErrorNumber);
         status.Append(": ");
         status.Append(lic.LastError.ToString());
         break;
   }

   return status.ToString();
}
```

VB.NET

```
Friend Shared Function GenerateLicenseErrorString(lic As License) As String
   Dim status As New StringBuilder()

   Select Case lic.LastError.ErrorNumber
```

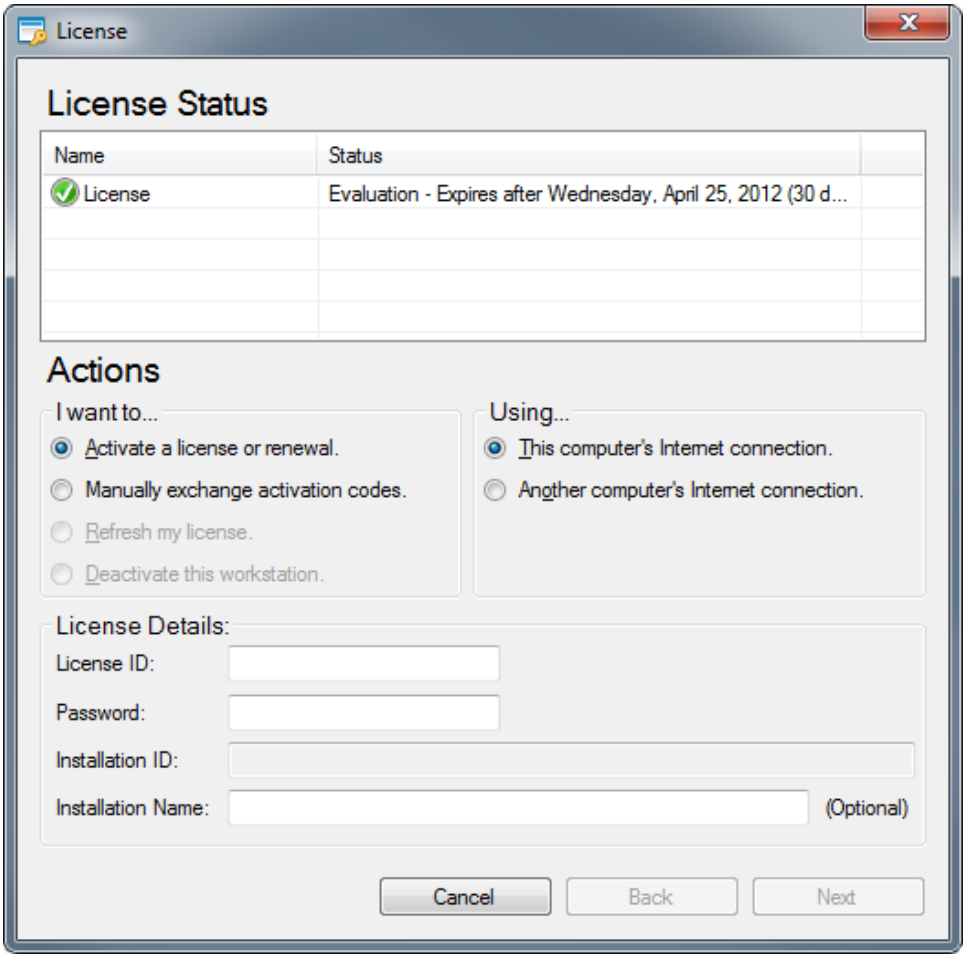
```
      Case LicenseError.ERROR_COULD_NOT_LOAD_LICENSE
         status.Append(lic.LastError.ErrorNumber)
         status.Append(": License not found - activation is required.")
      Case LicenseError.ERROR_WEBSERVICE_RETURNED_FAILURE 'Web service error message.
         status.Append(lic.LastError.ExtendedErrorNumber)
         status.Append(": ")
         status.Append(LicenseError.GetWebServiceErrorMessage(lic.LastEr-
         ror.ExtendedErrorNumber))
      Case Else 'Show a standard error message.
         status.Append(lic.LastError.ErrorNumber)
         status.Append(": ")
         status.Append(lic.LastError.ToString())
   End Select

   Return status.ToString()
End Function
```

Your application would then use this function for displaying any error messages displayed in the application's forms, as well as for any license status entries that display errors in PLUSManagedGui's license management form.

Customizing the Status in the License Management Form

The license management form in PLUSManagedGui is designed to display information about the status of the license in a list. Each item in the list can contain a status icon and customizable text. This gives you the ability to report on the details of the license being used, including information such as the status of individual modules or features, the number of network seats allowed and/or available, etc...



Creating a Basic Status Entry

An example excerpt of source code is provided below, which shows how to create and use a single, simple status entry that shows the license status.

C#

```
LicenseStatusIcon entryIcon;
StringBuilder status = new StringBuilder();

if (m_LastLicenseValidationResult)
{
   entryIcon = LicenseStatusIcon.Ok;
   status.Append("OK");
}
else
{
   entryIcon = LicenseStatusIcon.Error;
   status.Append(m_License.LastError.ToString());
}
```

```
licensingGui1.LicenseStatusEntries.Clear();
licensingGui1.LicenseStatusEntries.Add(new LicenseStatusEntry(entryIcon, "License",
status.ToString()));
```

VB.NET

```
Dim entryIcon As LicenseStatusIcon
Dim status As New StringBuilder()

If m_LastLicenseValidationResult Then
   entryIcon = LicenseStatusIcon.Ok;
   status.Append("OK");
Else
   entryIcon = LicenseStatusIcon.[Error]
   status.Append(m_License.LastError.ToString())
End If

licensingGui1.LicenseStatusEntries.Clear()
licensingGui1.LicenseStatusEntries.Add(New LicenseStatusEntry(entryIcon, "License",
status.ToString()))
```

Of course, it is possible to extend on this code to include any number of details about the license, such as the expiration date, the type of license, etc... This example code only uses two of the possible status icons that may be used.

Status Icons

A list of status icons that may be used in status entries is provided below.

Icon	Enumeration Value
	LicenseStatusIcon.None
	LicenseStatusIcon.Ok
	LicenseStatusIcon.Error
	LicenseStatusIcon.Information
	LicenseStatusIcon.Unavailable
	LicenseStatusIcon.Warning

PLUSManagedGui: Implementing Basic Processing

When PLUSManagedGui processes a user's request (such as an activation, license refresh, or deactivation), the LicenseManagementActionComplete event is fired to allow your application to react to the newly-processed request. This step is required to save any changes made to your application's license.

Creating the Event Handler

The first step to handling PLUSManagedGui processing results is creating the event handler. The handler method will need to accept a `LicenseManagementActionCompleteEventArgs` object for the event arguments (or second) parameter. The example code provided below shows how to create the method.

C#

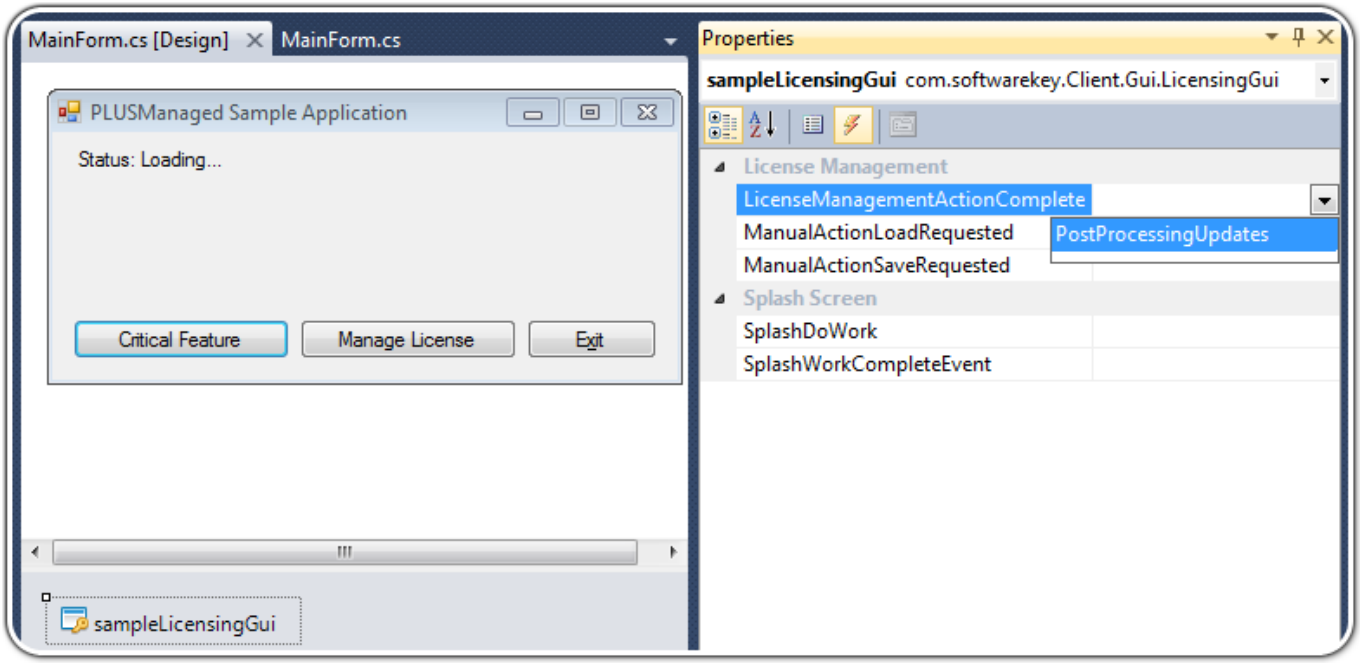
```
private void PostProcessingUpdates(object sender, LicenseManagementActionCompleteEventArgs e)

{
   //TODO: This is where your processing code needs to be added.
}
```

VB.NET

```
Private Sub PostProcessingUpdates(sender As Object, e As LicenseMan-
agementActionCompleteEventArgs)
   'TODO: This is where your processing code needs to be added.
End Sub
```

Once you have created the method as shown above, you then need to wire this method to the event. To do this, open your form in the designer, click on the `LicensingGui` object, click the lightning bolt icon in the properties pane, and select the newly created `PostProcessingUpdates` method in the drop-down menu for the `LicenseManagementActionComplete` event. This is shown in the screen shot below.



With the event handler now in place, the next task is to implement the processing code inside of the new method (in place of the TODO comments in the example code provided above).

Implementing Processing

Below is some example processing code which is designed to handle typical online and manual activation, license refresh, and deactivation requests. For the sake of simplicity, this code does not support downloadable and volume licenses, or trigger codes; however, the sample applications do include code that show you how to add support for these types of requests.

C#

```
private void PostProcessingUpdates(object sender, LicenseManagementActionCompleteEventArgs e)

{
   //TODO: If your application needs to support trigger codes, additional processing should
   be added here.

   if (LicenseError.ERROR_WEBSERVICE_RETURNED_FAILURE == e.LastError.ErrorNumber)
   {
      switch (e.LastError.ExtendedErrorNumber)
      {
         case 5010: //Invalid Product ID
         case 5015: //Invalid Installation ID
         case 5016: //Deactivated installation
         case 5017: //Invalid license status
            //The last action failed because the license is no longer valid in SOLO Server, so
            revoke/remove the license.
            m_License.RemoveLicense();
            m_LastLicenseValidationResult = false;
            break;
         default:
            //The web service returned an error, which should be shown to the user.
            e.PostProcessingSuccessful = false;
            break;
      }
   }
   else if (LicenseManagementActionTypes.OnlineDeactivation == e.ActionType ||
      LicenseManagementActionTypes.ManualDeactivation == e.ActionType)
   {
      m_License.RemoveLicense();
      m_LastLicenseValidationResult = false;
   }
   else if (e.ProcessedSuccessfully &&
      (LicenseManagementActionTypes.OnlineActivation == e.ActionType ||
      LicenseManagementActionTypes.OnlineRefresh == e.ActionType ||
      LicenseManagementActionTypes.ManualActivation == e.ActionType ||
      LicenseManagementActionTypes.ManualRefresh == e.ActionType))
   {
      //This was a successful activation or license refresh attempt, which means we have a new
      license to process...
      //Save the new license.
      if (!m_License.SaveLicenseFile(e.NewLicenseContent))
      {
```

```
        //The new license could not be saved.
        e.PostProcessingSuccessful = false;
        e.LastError = m_License.LastError;
        m_LastLicenseValidationResult = false;
        return;
      }

      //Now try to reload the new license.
      if (!m_License.LoadFile(LicenseFilePath))
      {
        //The new license could not be loaded.
        m_LastLicenseValidationResult = false;
        return;
      }

      //TODO: Perform any downloadable or volume license processing here.
    }

    m_LastLicenseValidationResult = m_License.Validate();
}
```

VB.NET

```
Private Sub PostProcessingUpdates(sender As Object, e As LicenseMan-
agementActionCompleteEventArgs)
   'TODO: If your application needs to support trigger codes, additional processing should be
   added here.

   If LicenseError.ERROR_WEBSERVICE_RETURNED_FAILURE = e.LastError.ErrorNumber Then
      Select Case e.LastError.ExtendedErrorNumber
         Case 5010, 5015, 5016, 5017 'Invalid Product ID, Invalid Installation ID, Deactivated
         installation, or Invalid license status, respectively.
            m_License.RemoveLicense()
            m_LastLicenseValidationResult = False
         Case Else
            'The web service returned an error, which should be shown to the user.
            e.PostProcessingSuccessful = False
            e.LastError = m_License.LastError
      End Select
   ElseIf LicenseManagementActionTypes.OnlineDeactivation = e.ActionType OrElse LicenseMan-
   agementActionTypes.ManualDeactivation = e.ActionType Then
      m_License.RemoveLicense()
      m_LastLicenseValidationResult = False
   ElseIf e.ProcessedSuccessfully AndAlso (LicenseManagementActionTypes.OnlineActivation =
   e.ActionType OrElse LicenseManagementActionTypes.OnlineRefresh = e.ActionType OrElse
   LicenseManagementActionTypes.ManualActivation = e.ActionType OrElse LicenseMan-
   agementActionTypes.ManualRefresh = e.ActionType) Then
      'This was a successful activation or license refresh attempt, which means we have a new
      license to process.
      'Save the new license.
      If Not m_License.SaveLicenseFile(e.NewLicenseContent) Then
         'The new license could not be saved.
         e.PostProcessingSuccessful = False
         e.LastError = m_License.LastError
         m_LastLicenseValidationResult = False
         Return
      End If

      'Now try to reload the new license.
      If Not m_License.LoadFile(LicenseFilePath) Then
         'The new license could not be loaded.
         m_LastLicenseValidationResult = False
         Return
      End If

      'TODO: Perform any downloadable or volume license processing here.
   End If

   m_LastLicenseValidationResult = m_License.Validate()
End Sub
```

This example code covers the following scenarios:

- If an action is being performed online via SOLO Server, and an error occurs, the error is displayed to the user in the license management form.

- If SOLO Server returns a result code indicating the license or activation is no longer valid, the license is revoked or removed from the system.
- If the user has deactivated, the license is revoked or removed from the system.
- If an activation or license refresh has been processed successfully, the new license file is saved, loaded, and validated.

PLUSManagedGui: Delayed Request Processing

Delayed request processing can be necessary when a customer needs to activate his or her computer using another computer's Internet connection, or by using trigger codes. By saving information about the request being processed to a file (or another location of your choice), the request may be processed at a later time, even if the application has been closed. Delayed request processing is especially useful when allowing activation via phone, fax, or email.

Define Configuration Properties

Before you implement the code that saves and loads the manual request data, it is best to first define some configuration properties for the location where the data will be saved, as well as the key that will be used when encrypting and decrypting the data.

Defining a Location

When selecting a location for saving the manual request information, there are several considerations to make:

- It is typically best to pick a user-specific location to avoid issues with permissions, as this avoids issues with the User Access Control (or UAC) feature in Windows Vista and later. However, picking a user-specific location means that if a user opts to resume at a later time, only that user would be able to complete the process with a response to that request. If another user attempts activate the same system, a completely new request would need to be generated for that other user.
- The location needs to be unique to the application being licensed. If you are licensing something that acts as a module, add-in, or plug-in to another application, you need to ensure that the location selected is unique to what you are licensing (so that it does not conflict with other software run in this same environment).

The example code provided below shows how to define a location unique to your application (or .EXE file) in a user-specific location.

C#

```
internal static string ManualActionSessionStateFilePath
{
   get
   {
      return Path.Combine(Environment.GetFolderPath(Environment.Spe-
      cialFolder.LocalApplicationData),
         Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().Location) + "Manu-
         alAction.xml");
   }
}
```

VB.NET

```
Friend Shared ReadOnly Property ManualActionSessionStateFilePath() As String
   Get
      Return Path.Combine(Environment.GetFolderPath(Environment.Spe-
      cialFolder.LocalApplicationData), _
         Path.GetFileNameWithoutExtension(Assembly.GetExecutingAssembly().Location) & "Manu-
         alAction.xml")
   End Get
End Property
```

Although the example code above uses a file, your application may save this data wherever it may be most appropriate. This can, for example, include locations such as an application database, if one is used.

Defining Encryption Key Data

Encrypting the manual request data is important for preventing the data from being updated by users arbitrarily (which can lead to issues and concerns such as replay attacks). Before you can use encryption in your code, you will need to define what algorithm will be used, and what keys will be used with the algorithm. Below is some very basic code that shows how to define these keys for the example code provided later in this topic.

C#

```
internal static string ManualActionIV
{
   get { return "IZWGSERAx1h9zzpCGBmRXg=="; }
}

internal static string ManualActionKey
{
   get { return "ajOsfy2RCC66qD5D9bCiJIK5zJVYUbJwLXwwC6fW4SI="; }
}
```

VB.NET

```
Friend Shared ReadOnly Property ManualActionIV() As String
   Get
      Return "IZWGSERAx1h9zzpCGBmRXg=="
   End Get
End Property

Friend Shared ReadOnly Property ManualActionKey() As String
   Get
      Return "ajOsfy2RCC66qD5D9bCiJIK5zJVYUbJwLXwwC6fW4SI="
   End Get
End Property
```

Important

The encryption keys used in the example code shown here and in the sample applications should **not** be used in your application! Your application needs to use its own, unique encryption keys to be secure.

Additionally, the encryption keys should be stored in a obfuscated fashion, and not as a string as shown above. This is because, if left as a string literal, a hacker may be able to find the string very easily. Most obfuscation tools help with this, but it is still best to break this data apart and hide it in different areas of your application's code.

Saving the Request Data

First, you must create a method for saving the data. It is important that the data is encrypted by this method to help prevent users from arbitrarily updating the saved request data. Below is some example code that defines the method used to save the data.

C#

```
private void ManualActionSaveRequested(object sender, ManualActionSaveRequestedEventArgs e)
{
   string encryptedString = "";
   try
   {
```

```
      byte[] dataBytes = Encoding.UTF8.GetBytes(e.Contents);

      using (RijndaelManaged algorithm = new RijndaelManaged())
      using (ICryptoTransform encryptor = algorithm.CreateEncryptor(Convert.FromBase64String
      (ManualActionKey), Convert.FromBase64String(ManualActionIV)))
      using (MemoryStream encryptedStream = new MemoryStream())
      using (CryptoStream cipherStream = new CryptoStream(encryptedStream, encryptor,
      CryptoStreamMode.Write))
      {
         //Write all data to the crypto stream and flush it.
         cipherStream.Write(dataBytes, 0, dataBytes.Length);
         cipherStream.FlushFinalBlock();

         //Get the encrypted byte array.
         encryptedString = Convert.ToBase64String(((MemoryStream)encryptedStream).ToArray());
      }

      File.WriteAllText(ManualActionSessionStateFilePath, encryptedString);
   }
   catch (Exception)
   {
      //Do nothing if we cannot save the file.
   }
}
```

VB.NET

```
Private Sub ManualActionSaveRequested(sender As Object, e As Manu-
alActionSaveRequestedEventArgs)
   Dim encryptedString As String = ""
   Try
      Dim dataBytes As Byte() = Encoding.UTF8.GetBytes(e.Contents)

      Using algorithm As New RijndaelManaged()
         Using encryptor As ICryptoTransform = algorithm.CreateEncryptor
         (Convert.FromBase64String(ManualActionKey), Convert.FromBase64String(ManualActionIV))

            Using encryptedStream As New MemoryStream()
               Using cipherStream As New CryptoStream(encryptedStream, encryptor,
               CryptoStreamMode.Write)
                  'Write all data to the crypto stream and flush it.
                  cipherStream.Write(dataBytes, 0, dataBytes.Length)
                  cipherStream.FlushFinalBlock()

                  'Get the encrypted byte array.
                  encryptedString = Convert.ToBase64String(DirectCast(encryptedStream,
                  MemoryStream).ToArray())
               End Using
            End Using
         End Using
      End Using
      'Save the file to disk.
      File.WriteAllText(ManualActionSessionStateFilePath, encryptedString)
   Catch ex As Exception
      'Do nothing if we cannot save the file.
   End Try
End Sub
```

With the method created, the next step is to open your form's designer, select the `LicensingGui` component, click the lightning bolt icon in the properties pane, and select the newly-created method for the `ManualActionSaveRequested` event.

Loading the Request Data

Next, you must create a method which loads the data. If you encrypted the saved data, then this method also needs to decrypt the data that is loaded. This is shown in the example code below.

C#

```
private void ManualActionLoadRequested(object sender, ManualActionLoadRequestedEventArgs e)
{
   string data = ""; try
   {
      //Load the data needed to restore the manual action's state, which allows the response
      to be processed after the application has been closed.
      data = File.ReadAllText(ManualActionSessionStateFilePath);

      //Delete the file after we have loaded it.
      File.Delete(ManualActionSessionStateFilePath);
```

```
      byte[] dataBytes = Convert.FromBase64String(data);
      using (RijndaelManaged algorithm = new RijndaelManaged())
      using (ICryptoTransform decryptor = algorithm.CreateDecryptor(Convert.FromBase64String
      (ManualActionKey), Convert.FromBase64String(ManualActionIV)))
      using (MemoryStream decryptedStream = new MemoryStream(dataBytes))
      using (CryptoStream cipherStream = new CryptoStream(decryptedStream, decryptor,
      CryptoStreamMode.Read))
      {
         //Read the data out of the crypto stream.
         byte[] decryptedValue = new byte[dataBytes.Length];
         cipherStream.Read(decryptedValue, 0, decryptedValue.Length);

         StringBuilder plain = new StringBuilder();
         foreach (char character in Encoding.UTF8.GetChars(decryptedValue))
            plain.Append(character);
         data = plain.ToString();
      }
   }
   catch (Exception)
   {
      data = "";
   }

   e.Contents = data;
}
```

VB.NET

```
Private Sub ManualActionLoadRequested(sender As Object, e As Manu-
alActionLoadRequestedEventArgs)
   Dim data As String = ""
   Try
      'Load the data needed to restore the manual action's state, which allows the response to
      be processed after the application has been closed.
      data = File.ReadAllText(ManualActionSessionStateFilePath)

      'Delete the file after we have loaded it.
      File.Delete(ManualActionSessionStateFilePath)

      Dim dataBytes As Byte() = Convert.FromBase64String(data)
      Using algorithm As New RijndaelManaged()
         Using decryptor As ICryptoTransform = algorithm.CreateDecryptor
         (Convert.FromBase64String(ManualActionKey), Convert.FromBase64String(ManualActionIV))

            Using decryptedStream As New MemoryStream(dataBytes)
               Using cipherStream As New CryptoStream(decryptedStream, decryptor,
               CryptoStreamMode.Read)
                  'Read the data out of the crypto stream.
                  Dim decryptedValue As Byte() = New Byte(dataBytes.Length - 1) {}
                  cipherStream.Read(decryptedValue, 0, decryptedValue.Length)

                  Dim plain As New StringBuilder()
                  For Each character As Char In Encoding.UTF8.GetChars(decryptedValue)
                     plain.Append(character)
                  Next
                  data = plain.ToString()
               End Using
            End Using
         End Using
      End Using
   Catch ex As Exception
      data = ""
   End Try

   e.Contents = data
End Sub
```

With the method created, the next step is to open your form's designer, select the `LicensingGui` component, click the lightning bolt icon in the properties pane, and select the newly-created method for the `ManualActionLoadRequested` event.

PLUSManagedGui: Deploying Protected Applications

Review the **system requirements** for details on what is required to run PLUSManagedGui. Since PLUSManagedGui depends on PLUSManaged, be sure to read about **deploying applications protected with PLUSManaged**.

Distributing the PLUSManagedGui Library

The PLUSManagedGui should be installed in the same directory as your application (or .exe file). This greatly simplifies deployment and also prevents your application from using other versions the library that may not be compatible with your application (which could be installed by other applications that use PLUSManagedGui).

Important

Although installing the library to the application directory is **strongly** recommended, it is possible to install the assembly to the **Global Assembly Cache** (GAC) if required by your application.

If your application requires you to install dependencies in the GAC, make sure your application's reference to any Protection PLUS 5 SDK libraries (such as PLUSManaged or PLUSManagedGui) has the *Specified Version* property set to true.

Important

The Protection PLUS 5 SDK .NET libraries (such as PLUSManaged and PLUSManagedGui) are obfuscated, but this alone cannot prevent a hacker from modifying your application's source code. When your .NET application is compiled, it is compiled to what is called Microsoft Intermediate Language (MSIL - pronounced like "missile") code. There are tools that can be used to translate and reverse engineer MSIL code, so it is imperative that you select and use an obfuscation tool to further protect your application and your investments in it. Microsoft offers a **good summary** of this subject.

Assembly Embedding

It is possible to obfuscate your application in a way which embeds the Protection PLUS 5 SDK libraries with your application. This is recommended where possible, as it works with PLUSManaged and PLUSManagedGui, provides some extra security with most obfuscation tools, and can help simplify your deployment (as won't be necessary to distribute the library once it is embedded in your application).

PLUSNative Overview

If your application is a native application (written in C/C++, for example), or is written in a language which can make calls to shared/dynamic-link libraries, then PLUSNative is the API designed for your needs (see more about **selecting a solution**).

The purpose of this section is to guide developers through the process of using the PLUSNative API in applications. Understanding this portion of the manual and using the API effectively requires at least a basic understanding of application development and procedural programming, and also requires a basic understanding of the **SoftwareKey System™** and **licensing**.

Important

The steps laid out in the common implementation sub-topics of this manual do not match exactly what is in the **Protection PLUS 5 SDK sample projects**. The manual topics are more simplified to make it easier to explain, while the sample projects have more complex organization, but the resulting functionality is similar in both cases.

Calling Conventions

Calling conventions define how functions or subroutines receive parameter data from the calling code, as well as how the a result is returned. The default calling convention used by your application can vary on operating system, processor architecture, and the programming language used. If you are using an include file provided with PLUSNative (such as PLUSNative.h for C/C++, PLUSNative.bas for Visual Basic 6, PLUSNative.pas for Delphi), then the calling conventions will already be specified correctly for your application. However, if you find you need to define the function prototypes for calling PLUSNative functions, then using the correct calling convention is imperative. The calling conventions used are as follows:

- 32 bit (x86) versions of Windows use `stdcall`.
- 32 bit (x86) versions of Linux and OS X use `cdecl`.
- 64 bit (x86_64) versions of Windows, Linux, and OS X use `fastcall`.

Include Files

The Protection PLUS 5 SDK installation directory has a sub-directory named *Include*. Each file in this directory contains definitions your application needs to call PLUSNative, and in many cases, it may also contain helper functions that simplify the use of PLUSNative by addressing language-specific limitations and nuances. (An example if this is how the Visual Basic 6 PLUSNative.bas file has its own implementation for `SK_ApiResultCodeToString`, so you do not need to worry about how to properly deal with string buffers allocated by our library in your code.)

Library Files

The Protection PLUS 5 SDK installation directory also contains a sub-directory named *Library*. Here, you will find a variety of builds of the PLUSNative API, which are organized as outlined below.

Dynamic Link Libraries / Shared Objects

Dynamic Link Libraries (DLLs), or shared objects, are organized with the following structure: *Operating System/[DLL/shared]/Architecture*. So for example, the 64 bit Windows DLL is located in *Library\Windows\DLL\x64*, and the universal OS X library is located in *Library/macOS/shared/universal*. When used, you will need to **distribute** one or more library files with your protected application(s).

Static Libraries

Static libraries allow you use PLUSNative with your application(s) without the need to **distribute** additional library files (this only applies to languages that can statically link a C library, such as C/C++). These files are organized with the following structure: *Operating System/Development Environment/[import/static/static-nodeps]/Architecture*. The third-level of this structure describes the nature of the static libraries, which include:

- import: These are import libraries that may be linked when using Dynamic Link Libraries (DLLs)/Shared Objects.
- static: These are static libraries which include third-party dependencies used by PLUSNative, such as OpenSSL, libcurl, and libxml2.
- static-nodeps: These are static libraries which do not include third-party dependencies used by PLUSNative. There are some cases (unrelated to the use of PLUSNative) where you might need to link your own dependencies, such as if you need to use a different version, or if you use a custom patch to resolve some issue your application experienced with the dependency. For these niche scenarios, the libraries in static-nodeps may be used, but leave you responsible for linking all of the required dependencies (OpenSSL, libcurl, and libxml2).

Next, you might also notice the static libraries for Windows include two versions named PLUSNative.lib and PLUSNative_md.lib. This is related to the *Runtime Library* setting in Visual C++ projects (under *Configuration Properties/C++/Code Generation* in the project properties). The PLUSNative.lib file may be used with the "Multi-threaded (/MT)" and "Multi-threaded Debug (/MTd)" runtime libraries, while the PLUSNative_md.lib file may be used with the "Multi-threaded DLL (/MD)" and "Multi-threaded Debug DLL (/MDd)" runtime libraries.

Getting Acquainted with the PLUSNative API

There are a few things you need to know when implementing PLUSNative.

Understanding PLUSNative's I/O functions

PLUSNative has a variety of functions available to assist with common file input and output (I/O) tasks that are generally necessary for licensing, but are made available to you for convenience. The relevance of these convenience functions to PLUSNative is that they provide you a way to fully control all steps of reading and saving licensing data, while also helping you avoid potential conflict with PLUSNative in areas like character encoding, memory management, formatting requirements, etc... PLUSNative offers a few different types/levels of convenience functions to consider for reading and saving license data:

- High-level functions with built-in I/O includes functions such as `SK_PLUS_LicenseFileLoad`, `SK_PLUS_LicenseFileSave`, `SK_PLUS_LicenseAliasAdd`. These functions accept a path to files or registry keys (in Windows), and automatically determine what type of path was provided. These functions will also automatically change permissions (unless you opt-out by specifying the global `SK_FLAGS_PERMISSIONS_NEVER_AUTO_SET` flag in the **context**).
- High-level functions without built-in I/O includes functions such as `SK_PLUS_LicenseLoad`, `SK_PLUS_LicenseGetXmlDocument`, `SK_PLUS_LicenseGetDocumentString`. When combined with the XML Encryption functions provided by PLUSNative, these can be leveraged to work with licenses that are loaded and saved using your own I/O routines. This gives you full control over where and how license data is stored, thus allowing you to store license data in custom locations (such as in a database, a custom hardware location, etc...).
- XML Encryption functions includes functions like `SK_XmlDocumentEncryptRsa` and `SK_XmlDocumentDecryptRsa`, which can be used for accessing data outside of higher-level functions, implementing custom I/O, and implementing customized web service requests.
- Low-level access functions includes functions such as `SK_FileReadAllText`, `SK_FileWriteAllText`, `SK_RegistryValueGet`, `SK_RegistryValueSet`. These functions only accept the type of path as indicated in the function name (e.g. `SK_FileReadAllText` does not accept a registry path), they do not perform any other special tasks automatically (such as setting permissions). However, they provide a simple way of retrieving string data from a file or registry key that may be used in further calls to PLUSNative functions. For example, this could include things like loading manual activation data from an unfinished activation attempt, retrieving some information for use in a custom identifier, etc...

Working with Windows Registry Paths

When working in Microsoft Windows, you may opt to store the license file and/or alias files in the registry. To use registry paths in PLUSNative that support them, you will need to specify a special prefix to the path string to indicate the hive you wish to access as follows:

Hive	Prefix
HKEY_CURRENT_USER	:CU:
HKEY_LOCAL_MACHINE	:LM:
HKEY_CLASSES_ROOT	:CR:

Additionally, the name of the value must be appended to the end of the path, after a colon (:) character. So for example, to access a value named *MyValue* under a *HKEY_LOCAL_MACHINE\Software\MyCompany\MyProgram* key, you would provide PLUSNative a path like: `":LM:Software\MyCompany\MyProgram:MyValue"`.

Strings, Encoding, and Locale

As a C API, PLUSNative uses **null-terminated strings**. By default, PLUSNative relies on a neutral locale, and expects all strings to use UTF-8 encoding. Additionally, PLUSNative dynamically allocates strings so they only use the amount of space that is required. The benefit is that there is no need to worry about allocating a buffer with enough space, reallocating buffers when there is not enough space, and it tends to be much easier to work with when using higher-level programming languages and frameworks. However, the downside is that you are responsible for freeing the memory allocated using the provided `SK_StringDispose` function. In most cases, sample code or language bindings provided will handle this for you reliably. However, this is definitely something to be well aware of when implementing PLUSNative using a language where bindings have not already been provided. See the memory management section below for additional details.

Memory Management

Since PLUSNative provides a C API, it is necessary for memory to be managed correctly. In general, safest approach is to base your code off our sample applications. This is because the sample code (which is generally written in higher-level languages, like C++, Objective C, etc...) already manages memory for you correctly. However, if you are implementing PLUSNative from scratch, then it's pivotal that you understand how to manage memory correctly with this API. Failing to do so can result in memory leaks, random crashes, and can leave your users with a general experience of instability with your application(s). The important things you must understand to avoid these kinds of issues are outlined at a high level below.

Initialization Functions

In PLUSNative, initialization functions can include any function that allocates memory for an object which is returned. This can include things such as the API Context, a string, an XML document, etc...

Disposal Functions

When your application is done using some object or data that PLUSNative has provided/returned, you must use PLUSNative to de-allocate the memory that was previously allocated from an initialization function.

Understand Best Practices

Proper memory management is a critical area of concern for programmers, as it can affect application performance, stability, and (perhaps most importantly) security. Preventing leaks and memory corruption is essential to providing your users a responsive, reliable, and safe experience with your applications. Some practices are outlined below that can be helpful in general, and are essential when using PLUSNative.

Initialize Variable Values

Any time you declare a variable, especially a pointer or an array, initialize the data. (Generally, this data should be initially zero.) Some examples include:

- Initialize buffer/array elements to `0` -- especially character arrays. For example: `char mychar[10] = {0};` will initialize all array elements to 0, making mychar an empty string intially. (If you allocate memory on the heap, C/C++ applications can use the `ZeroMemory` macro [Windows only] or the `memset` function to zero the memory after the declaration.) Doing this helps ensure that anything that might be called on this string (such as `strlen`) before the value is initialized with a meaningful string will not run over the bounds of the buffer allocated.
- Initialize numeric values to `0` or the most appropriate value for the variable's use. This is especially important for anything used for Boolean expressions, where `0` valuates to `false`, and any other value evaluates to `true`. This can help ensure that, should the value not be initialized due to some unexpected condition, your application behaves in the safest way possible.
- Always initialize pointers to `NULL` (or `0`) at declaration. This prevents you from accidentally attempting to free memory that was never really allocated later in code (should you attempt to free memory with an uninitialized pointer). All PLUSNative initialization functions require your pointer to be initialized this way, so as to reject uninitialized pointers that may point to invalid memory addresses, or pointers that point to addresses which must first be freed before creating a new allocation (which would otherwise cause memory leaks).

Verify Pointers

First, consistently initializing pointers to `NULL` (as described above) is a critical part of verifying pointers reliably. With that practice firmly in place, you can always check that a pointer is not `NULL` before attempting to use it. Consider the following example excerpt of bad code:

C/C++

```
char *myString = NULL;
size_t myStringSize = 10;
myString = (char *)malloc(myStringSize);
memset(myString, 0, myStringSize); /*It is bad to do this without checking the pointer
first!*/
/*The program might have additional statements here, which could use myString.*/
free(myString); /*It is bad to do this without checking the pointer first!*/
```

Should the `malloc` function fail in the example above, the program would crash with an access violation in `memset` because `myString` would contain a `NULL` value. And because it is relatively unlikely for you to run into this condition while debugging and testing yourself, it is more likely that a customer would eventually report the crash (either on a very old computer, or possibly one affected by malware). This can easily be avoided by consistently using basic control structures as shown in the next example excerpt:

C/C++

```
char *myString = NULL;
size_t myStringSize = 10;
do
{
```

```
   myString = (char *)malloc(myStringSize);
   if (NULL == myString)
   {
      /*Your application can handle error reporting here.*/
      break;
   }
   memset(myString, 0, myStringSize);
   /*The program might have additional statements here, which could use myString.*/
} while (0);
if (NULL != myString)
{
   free(myString);
   myString = NULL;
}
```

Using the do/while control structure allows us to check pointers and conveniently skip any logic that would otherwise attempt to use an uninitialized pointer. Of course, you can use any control structure (such as try/catch) you feel is appropriate; what matters most is that you are consistent with your practices.) Additionally, in the expression that checks the pointer, the constant value is on the left side of the expression so as to avoid accidental assignment. (For example, `if (myString = NULL)` would assign a `NULL` value to `myString` and cause a memory leak with this example; and the expression would always evaluate to `FALSE`, causing the same exact problem as the first example.)

Free Memory Safely and Consistently

Freeing memory safely is directly related to the previous two points on consistently initializing pointers and verifying pointers before use. Attempting to free memory using an uninitialized pointer is just as problematic as attempting to use uninitialized pointers. Additionally, using control structures as shown above is critical to ensuring there are no memory leaks. Consider the example below, which expands on the prior examples:

C/C++

```
char *myString = NULL;
size_t myStringSize = 10;
char mySecondString = NULL;
size_t mySecondStringSize = 20;
do
{
   myString = (char *)malloc(myStringSize);
   if (NULL == myString)
   {
      /*Your application can handle error reporting here.*/
      break;
   }
   memset(myString, 0, myStringSize);

   mySecondString = (char *)malloc(mySecondStringSize);
   if (NULL == mySecondString)
   {
      /*Your application can handle error reporting here.*/
      break;
   }
   memset(mySecondString, 0, mySecondStringSize);

   /*The program might have additional statements here, which could use myString and/or
```

```
   mySecondString.*/
} while (0);
if (NULL != myString)
{
   free(myString);
   myString = NULL;
}
if (NULL != mySecondString)
{
 free(mySecondString);
 mySecondString = NULL;
}
```

In the example above, the benefit of this overall approach in organizing code is that it can prevent memory leaks in unusual circumstances such as having a successful allocation for `myString`, but a failed allocation for `mySecondString` (because `myString` is still freed even when we break out of the do-while loop because `mySecondString`'s allocation failed).

Consistency is Key

The examples above are meant to provide high-level suggestions. As with many things, there are many ways to be right. Regardless of the specific control structures and variations from the suggestions and examples here, it is pivotal that you adhere to your practices strictly, and consistently.

PLUSNative: Features & Compatibility

The PLUSNative library has an extensive set of features, and some of these features are specific to or limited to specific operating systems.

Feature	Windows	macOS	Linux
General Features			
Copy protection	✔	✔	✔
Read-only and writable **license files**	✔	✔	✔
Automatically sets privileges on license files	✔	✔	✔
Activation & Electronic License Management (ELM)			
Online **activation**, **background checking**, and **deactivation**.	✔	✔	✔
Manual activation	✔	✔	✔
Manual activation via trigger codes	✔	✔	✔
SOLO Server Integration APIs	✔	✔	✔
System Identification Algorithms			
Computer name	✔	✔	✔
Hard disk volume serial	✔	✔	✔
Network Interface Card (NIC)	✔	✔	✔
Network Name	✔		
User Name	✔	✔	✔
Advanced Licensing Features			
Virtual machine guest environment detection	✔	✔	✔

Windows Terminal Services / Remote Desktop detection	✔		
Network floating licensing via semaphore files	✔		
Cloud-Controlled Floating Network Licensing using SOLO Server	✔	✔	✔
Volume licenses	✔	✔	✔
Downloadable licenses with trigger code activation	✔	✔	✔

Adding PLUSNative to your application

PLUSNative is a pure standard C library that should be compatible with most modern C/C++ compilers and integrated development environments (IDEs) capable of using shared (DLL) or static C libraries. Refer to the particular documentation for your toolset to determine if they support C libraries and how to call them.

Header Files

The *SoftwareKey/PLUSNative/inc* directory contains the 'include' files (header files and library definition files). These contains all of the function prototypes and constants that make up the PLUSNative API.

PLUSNative.h

C/C++ header file.

PLUSNative.bas

VB6 library definition file.

PLUSNative.pas

Delphi 7+ (Pascal) library definition file.

Library Files

The `SoftwareKey/PLUSNative/lib` directory contains all the PLUSNative library files. 32-bit library files are located in the *x86* subdirectory, while 64-bit library files are located in the *x64* subdirectory. Library files for a given platform are found in their respective folders. This includes the *linux*, *macOS* and *windows* subdirectories. Within each of these directories you will find the *shared-dll*, *static* and *static-nodeps* directories. The *shared-dll* directory contains the shared or dynamic-link libraries, while the *static* directory contains the static libraries (or archives). The *static-nodeps* directory contains static libraries that do not bundle the third-party libraries used by PLUSNative. Refer to the **Third-Party Dependencies** section below for details on when and how to use these libraries.

The following provides a description of the various library files provided:

Linux

libPLUSNative.a

The PLUSNative static library.

libPLUSNative.so

The PLUSNative shared library.

macOS X

libPLUSNative.a

The PLUSNative static library.

libPLUSNative.dylib

The PLUSNative shared library.

Windows

PLUSNative.dll

The PLUSNative dynamic-link library.

PLUSNative_dllimport.lib

The PLUSNative dynamic-link import library. Link with this to early-bind to PLUSNative.dll.

PLUSNative.lib

The PLUSNative static library. Uses static C Runtime for projects compiled with /MT.

PLUSNative_md.lib

The PLUSNative static library. Uses dynamic C Runtime for projects compiled with /MD.

Third-party Dependencies

PLUSNative uses several third-party open source libraries as it's core. This includes libcurl, libxml2 and OpenSSL (libcrypto and libssl). Refer to the **Third-Party Licenses** topic for details on the permissive open source licenses used by these libraries. These libraries are linked into the shared dynamic-link libraries and do not need to be distributed separately. The static libraries also include these third-party libraries. If you happen to use any of these third-party libraries in your own application and intend to use the PLUSNative static library you should use the libraries provided in the *static-nodeps* directory, as these do not include these third-party libraries. If you use one or more of these third-party libraries, but not all of them, you will need to start linking all of them or consider using the PLUSNative shared dynamic-link library instead.

Other System Library Dependencies

The following system libraries are dependencies that must be linked into your application when linking the PLUSNative static libraries.

Linux

libm

The standard math library.

OS X

CoreFoundation

The *CoreFoundation* framework.

CoreServices

The *CoreServices* framework.

DiskArbitration

The *Disk Arbitration* framework.

IOKit

The *IOKit* framework.

Security

The *Security* framework.

Windows

Crypt32.lib

The Windows Cryptography (CrpytoAPI) library.

Ws2_32.lib

The Windows Sockets 2 (Winsock) library.

Universal Libraries for macOS

OS X supports universal (or fat) binaries that combines objects for multiple architectures in a single binary executable or library file. The 32-bit and 64-bit libraries provided with PLUSNative can easily be combined into a single universal binary using the OS X **lipo command**.

The following example demonstrates combining the 32-bit and 64-bit static libraries into a universal library. These commands would be issued in the OS X Terminal or executed from a bash script.

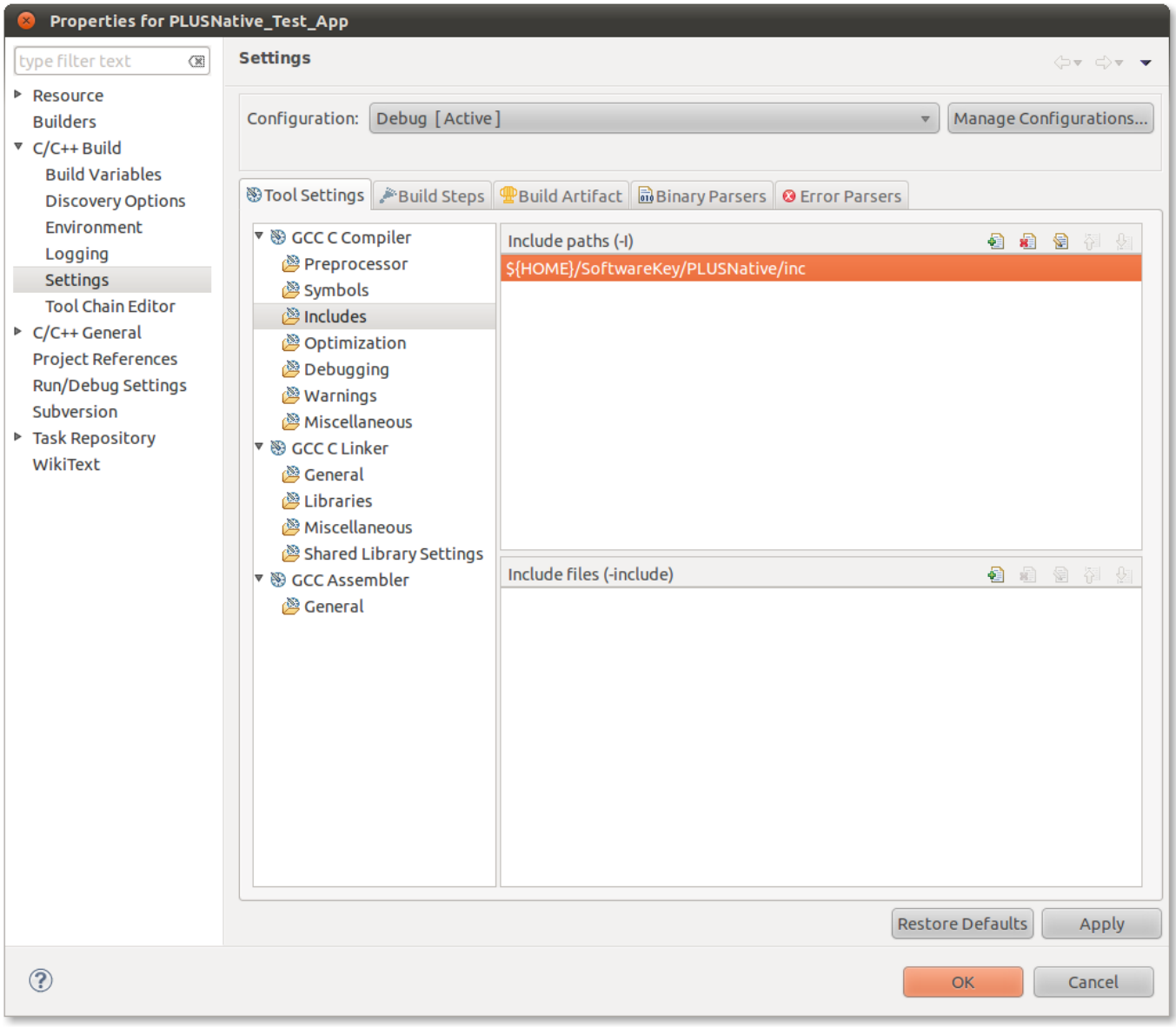
bash

```
cd ~/SoftwareKey/PLUSNative/lib
mkdir universal
lipo ./x86/macOS/static/libPLUSNative.a ./x64/macOS/static/libPLUSNative.a -create -output
./universal/libPLUSNative.a
```

Adding PLUSNative to Eclipse Project (Linux)

Include Search Paths

In order for the compiler to locate the *PLUSNative.h* header file that gets included with your source code you should add the *SoftwareKey/PLUSNative/inc* directory your Eclipse project's include search path. Start by opening up your project's properties page. This is done by right-clicking the project name in the Project Explorer and clicking **Properties**. Expand the *C/C++ Build* tree from the left and select **Settings**. Under the *GCC C Compiler* settings select **Includes**. Click the green plus (+) to the right of the *Include paths (-I)* section and add the path to the PLUSNative include directory (e.g. *SoftwareKey/PLUSNative/inc*).



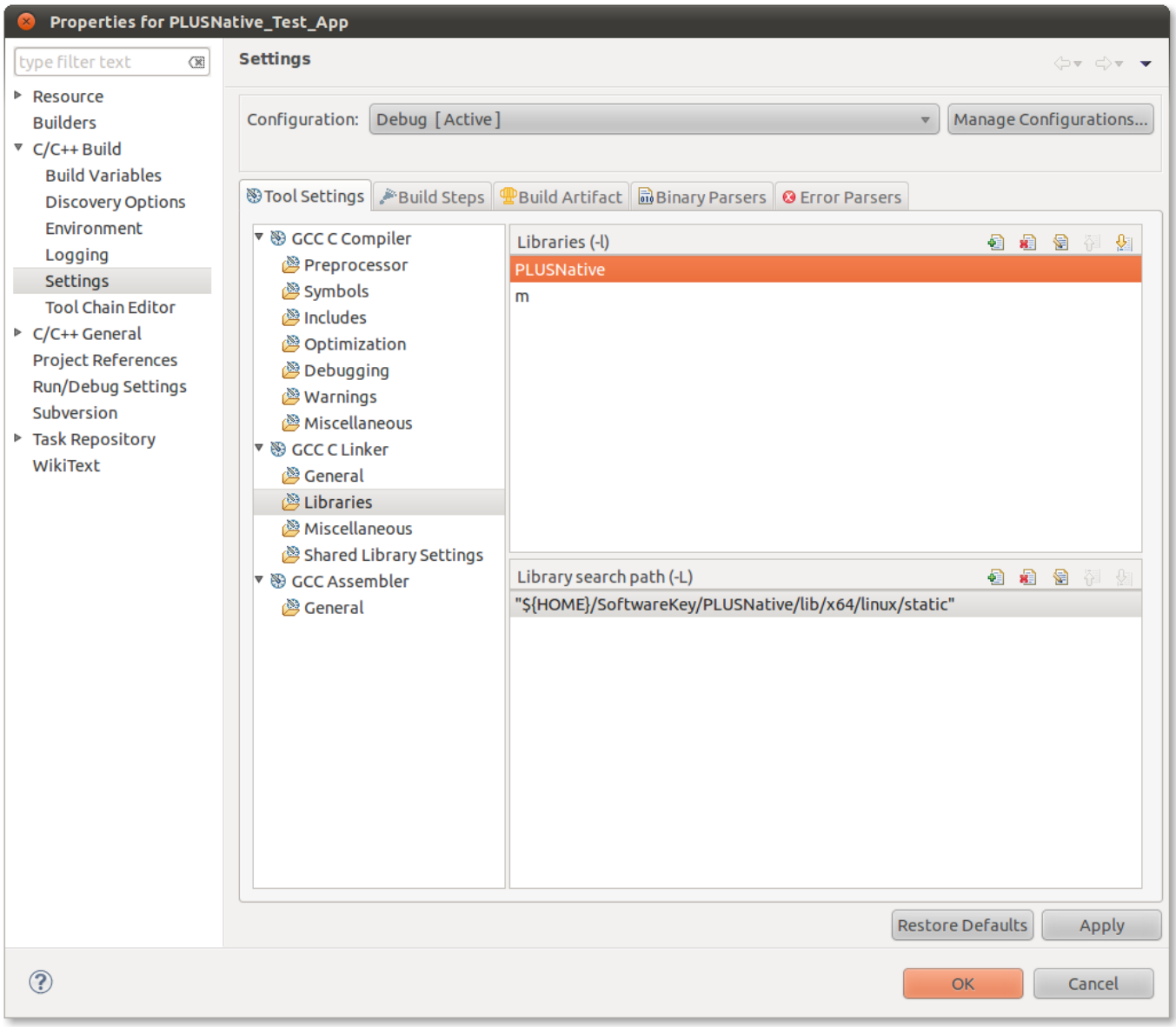
Library Search Paths and Library Dependencies

In order for the linker to locate the PLUSNative library file you will need to add the *SoftwareKey/PLUSNative/lib/...* directory your Eclipse project's library search path. Start by opening up your project's properties page. This is done by right-clicking the project name in the Project Explorer and clicking **Properties**. Expand the *C/C++ Build* tree from the left and select **Settings**. Under the *GCC C Linker* settings select **Libraries**. Click the green plus (+) to the right of the *Library search path (-L)* section and add the path to the PLUSNative library directory. The PLUSNative library directory will vary depending on whether you are using the 32-bit or 64-bit shared or static library. This example uses the 64-bit static library and sets the PLUSNative library search path to *SoftwareKey/PLUSNative/lib/x64/linux/static*.

After setting the library search path, you must tell the linker which libraries to link. Click the green plus (+) to the right of the *Libraries (-l)* section and add the PLUSNative library and any of it's **system library dependencies**.

Important

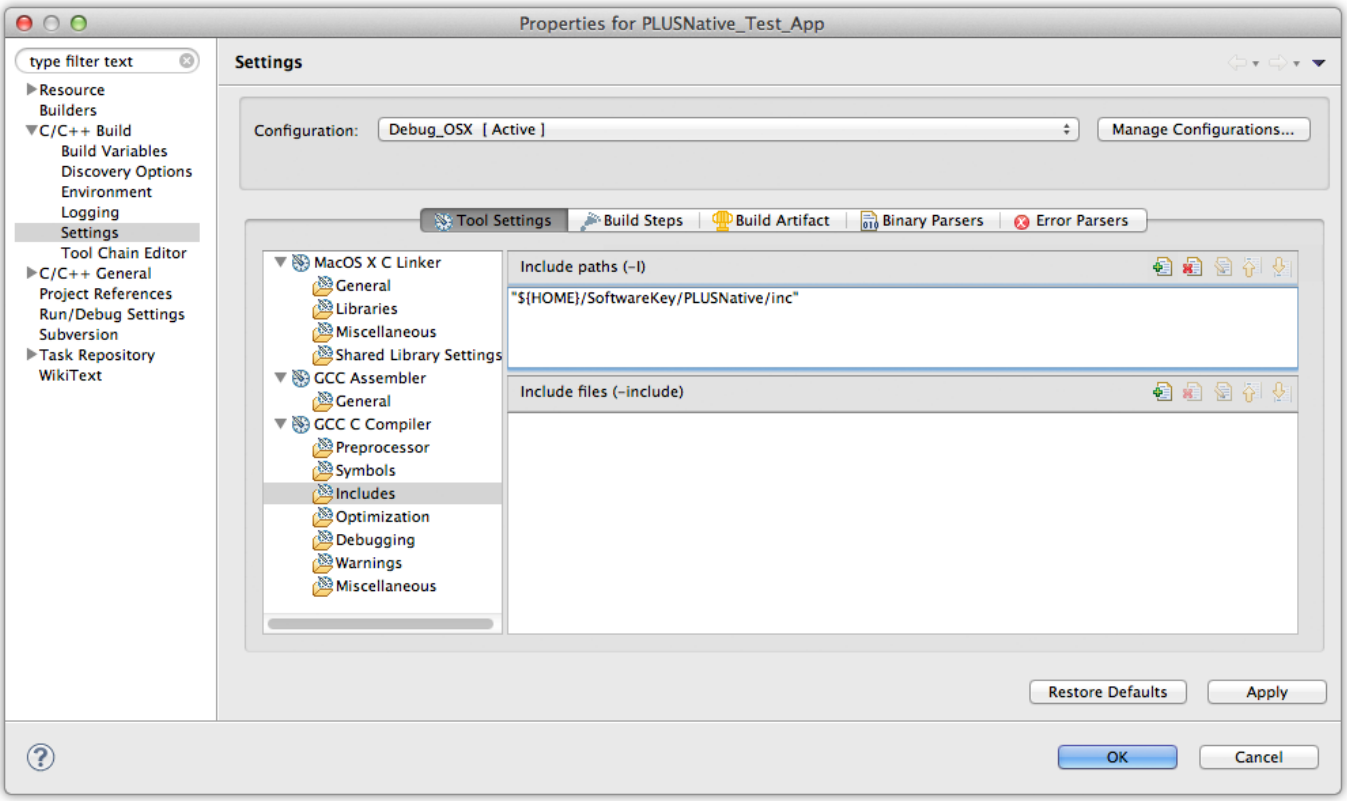
Do not include the "lib" prefix or the ".a" or ".so" suffix when adding a library. For example, instead of adding libPLUSNative.a, you would simply add PLUSNative.


Properties for PLUSNative_Test_App
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Logging
Settings
Tool Chain Editor
C/C++ General
Project References
Run/Debug Settings
Subversion
Task Repository
WikiText
Settings
Configuration: Debug [Active]
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
GCC C Compiler
Preprocessor
Symbols
Includes
Optimization
Debugging
Warnings
Miscellaneous
GCC C Linker
General
Libraries
Miscellaneous
Shared Library Settings
GCC Assembler
General
Libraries (-l)
PLUSNative
m
Library search path (-L)
"${HOME}/SoftwareKey/PLUSNative/lib/x64/linux/static"
Restore Defaults
Apply
OK
Cancel

Adding PLUSNative to Eclipse Project (macOS)

Include Search Paths

In order for the compiler to locate the *PLUSNative.h* header file that gets included with your source code you should add the *SoftwareKey/PLUSNative/inc* directory your Eclipse project's include search path. Start by opening up your project's properties page. This is done by right-clicking the project name in the Project Explorer and clicking **Properties**. Expand the *C/C++ Build* tree from the left and select **Settings**. Under the *GCC C Compiler* settings select **Includes**. Click the green plus (+) to the right of the *Include paths (-I)* section and add the path to the PLUSNative include directory (e.g. *SoftwareKey/PLUSNative/inc*).



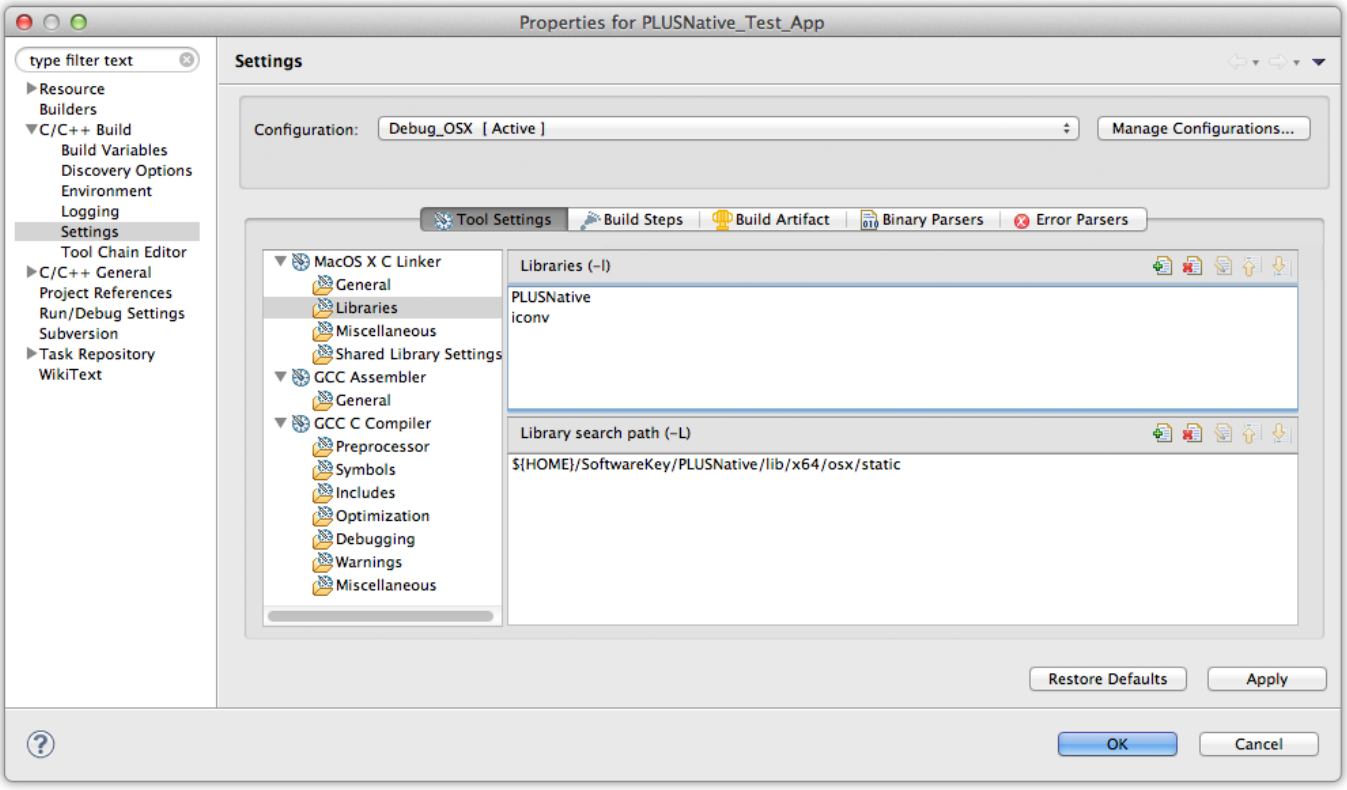
Library Search Paths and Library Dependencies

In order for the linker to locate the PLUSNative library file you will need to add the *SoftwareKey/PLUSNative/lib/...* directory your Eclipse project's library search path. Start by opening up your project's properties page. This is done by right-clicking the project name in the Project Explorer and clicking **Properties**. Expand the *C/C++ Build* tree from the left and select **Settings**. Under the *MacOS C Linker* settings select **Libraries**. Click the green plus (+) to the right of the *Library search path (-L)* section and add the path to the PLUSNative library directory. The PLUSNative library directory will vary depending on whether you are using the 32-bit or 64-bit shared or static library. This example uses the 64-bit static library and sets the PLUSNative library search path to *SoftwareKey/PLUSNative/lib/x64/macOS/static*.

After setting the library search path, you must tell the linker which libraries to link. Click the green plus (+) to the right of the *Libraries (-l)* section and add the PLUSNative library and any of it's **system library dependencies**.

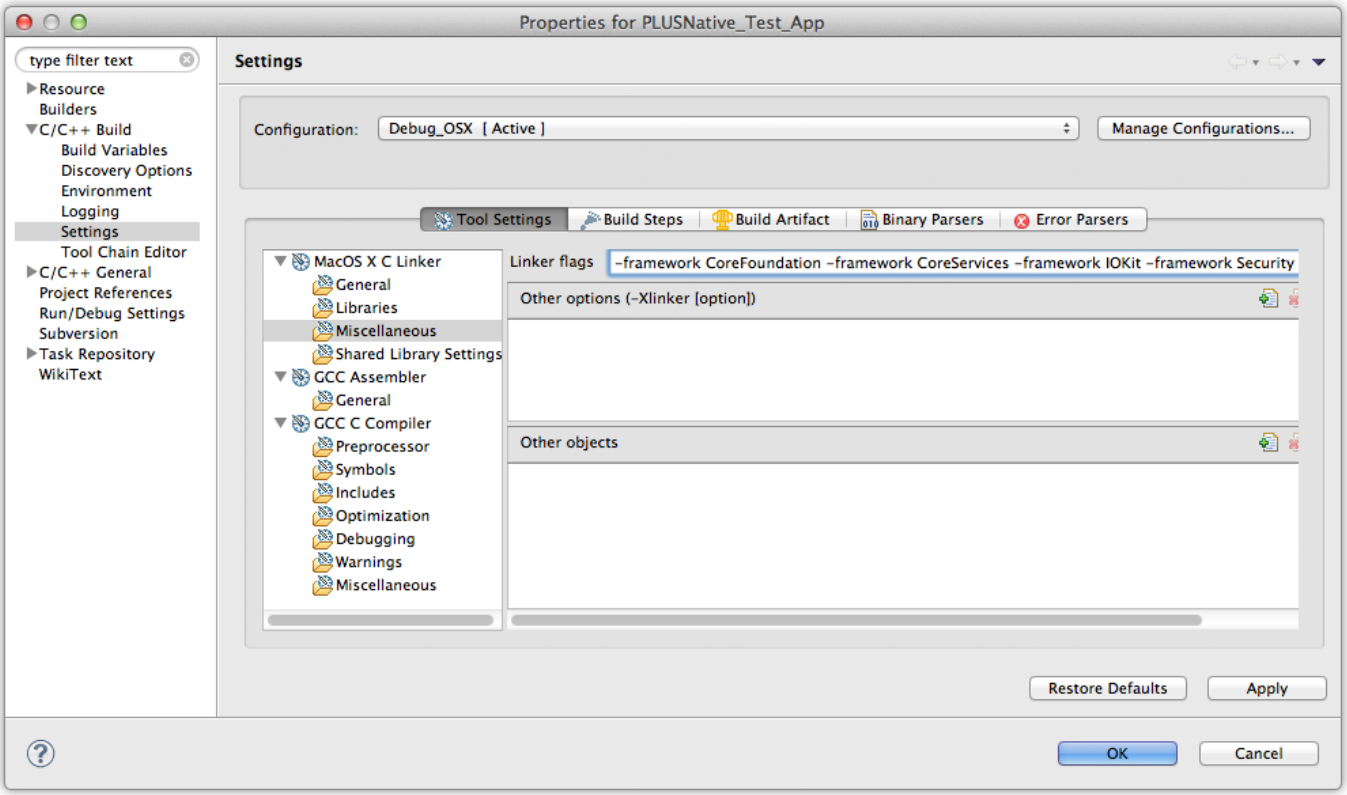
Important

Do not include the "lib" prefix or the ".a" or ".so" suffix when adding a library. For example, instead of adding libPLUSNative.a, you would simply add PLUSNative.



Framework Dependencies

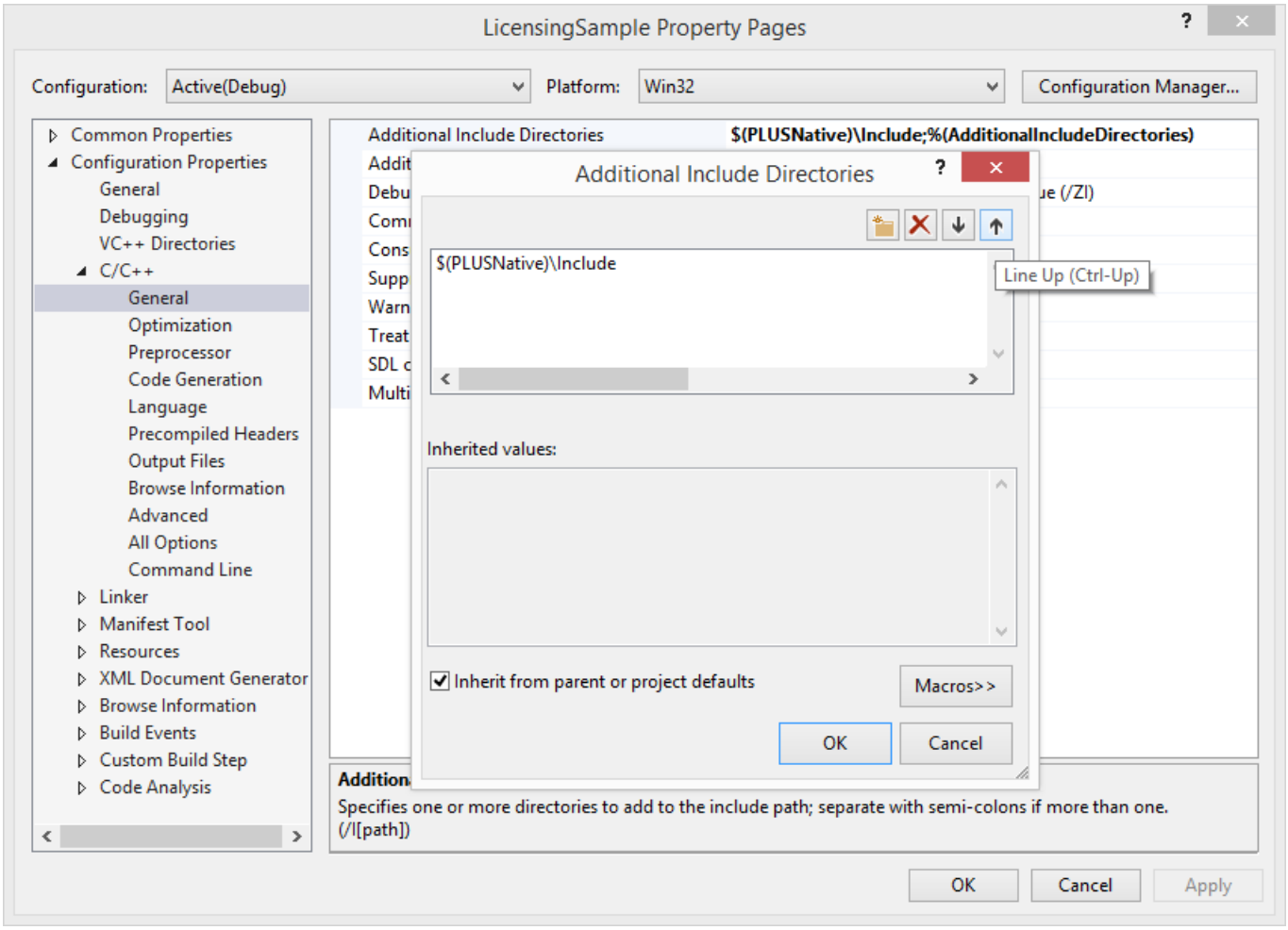
The PLUSNative library has several **system library dependencies** on OS X frameworks. Under the *MacOS X C Linker* settings select **Miscellaneous**. Add a "-framework [framework name]" flag to the *Linker flags* section for each framework required, replacing [framework name] with the actual name of the framework.


Properties for PLUSNative_Test_App
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Logging
Settings
Tool Chain Editor
C/C++ General
Project References
Run/Debug Settings
Subversion
Task Repository
WikiText
Settings
Configuration: Debug_OSX [Active]
Manage Configurations...
Tool Settings
Build Steps
Build Artifact
Binary Parsers
Error Parsers
MacOS X C Linker
General
Libraries
Miscellaneous
Shared Library Settings
GCC Assembler
General
GCC C Compiler
Preprocessor
Symbols
Includes
Optimization
Debugging
Warnings
Miscellaneous
Linker flags
-framework CoreFoundation -framework CoreServices -framework IOKit -framework Security
Other options (-Xlinker [option])
Other objects
Restore Defaults
Apply
OK
Cancel

Adding PLUSNative to Visual Studio Project

Include Search Paths

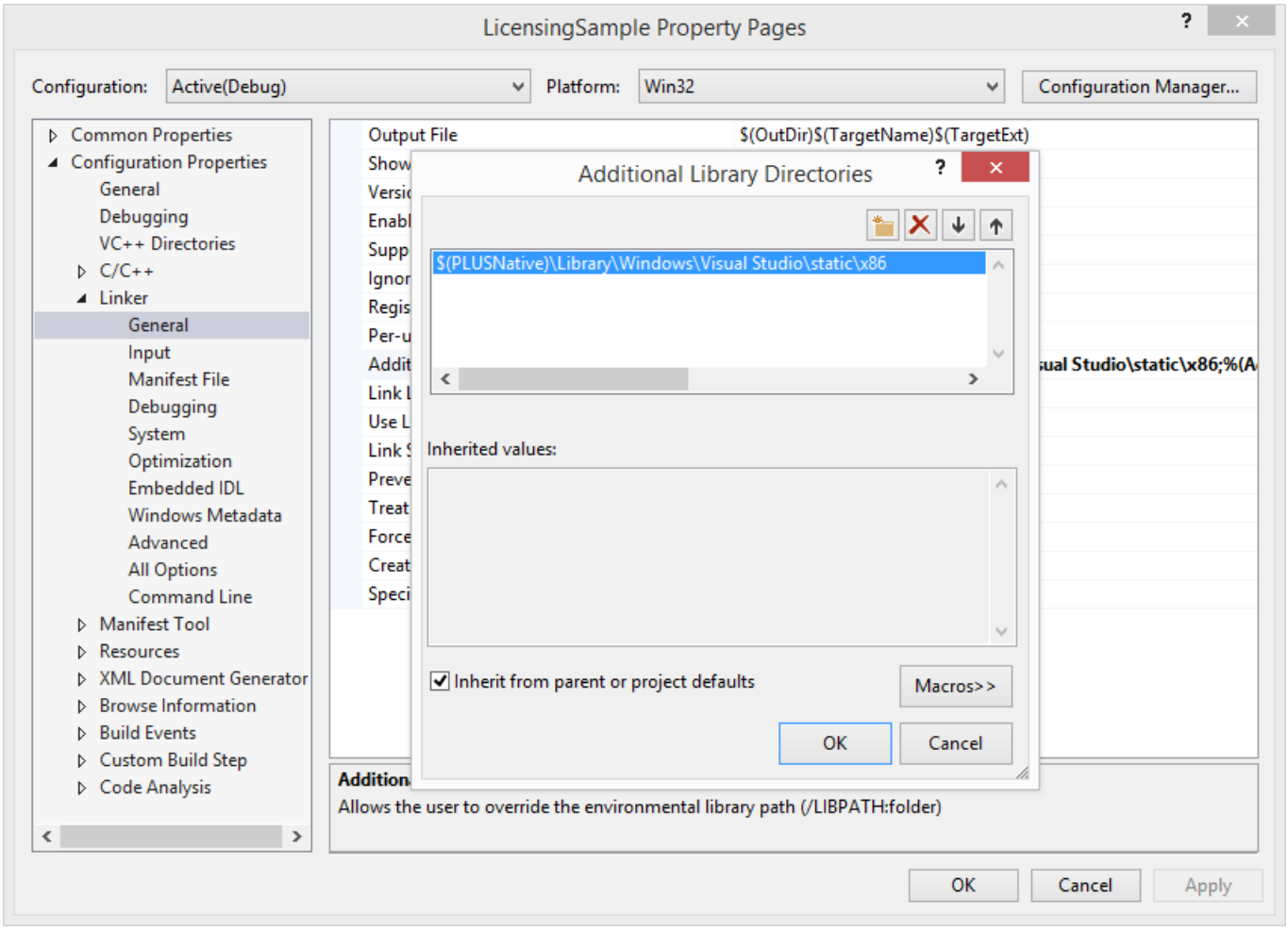
In order for the compiler to locate the *PLUSNative.h* header file that gets included with your source code you should add the *$(PLUSNative)\Include* directory your Visual Studio project's include search path. Start by opening up your project's properties page. This is done by right-clicking the project name in the Solution Explorer and clicking **Properties**. Expand the *C/C++* tree from the left and select **General**. Select the *Additional Include Directories* section, click the drop-down arrow to the right and click **<Edit...>**. Click the folder icon and add the path to the PLUSNative include directory (e.g. *$(PLUSNative)\Include*).



Library Search Paths

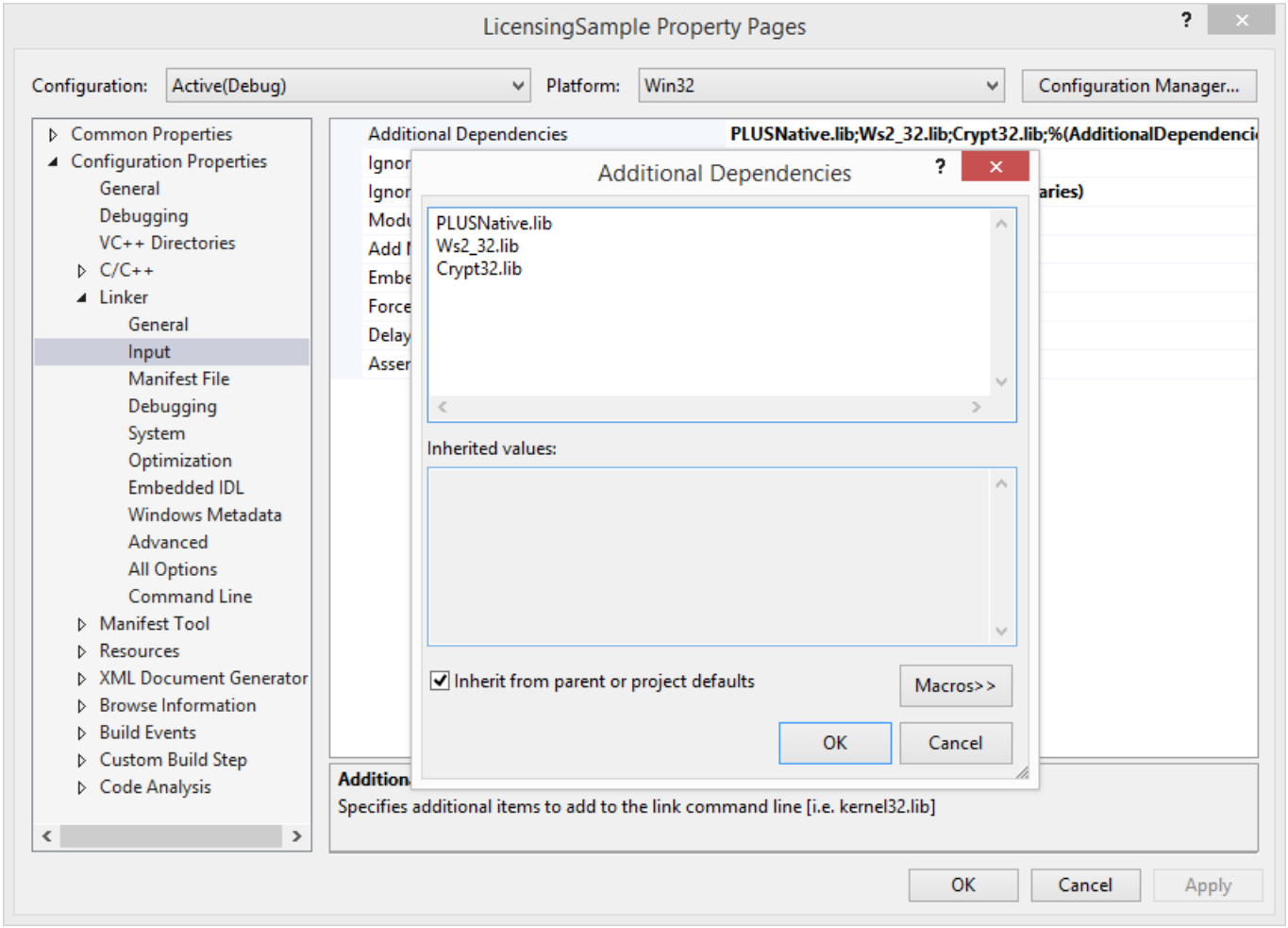
In order for the linker to locate the PLUSNative library file you will need to add the *$(PLUSNative)\Library\...* directory your Visual Studio project's library search path. Start by opening up your project's properties page. This is done by right-clicking the project name in the Solution Explorer and clicking **Properties**. Expand the *Linker* tree from the left and select **General**. Select the *Additional Library Directories* section, click the drop-down arrow to the

right and click **<Edit...>**. Click the folder icon towards the top right and add the path to the PLUSNative library directory. The PLUSNative library directory will vary depending on whether you are using the 32-bit or 64-bit shared or static library. This example uses the 32-bit static library and sets the PLUSNative library search path to *$(PLUSNative)\Library\Windows\Visual Studio\static\x86*.



Library Dependencies

After setting the library search path, you must tell the linker which libraries to link. Start by opening up your project's properties page. This is done by right-clicking the project name in the Solution Explorer and clicking **Properties**. Expand the *Linker* tree from the left and select **Input**. Select the *Additional Dependencies* section, click the drop-down arrow to the right and click **<Edit...>**. Click the folder icon towards the top right and add the PLUSNative library you wish to link and any of it's **system library dependencies**.



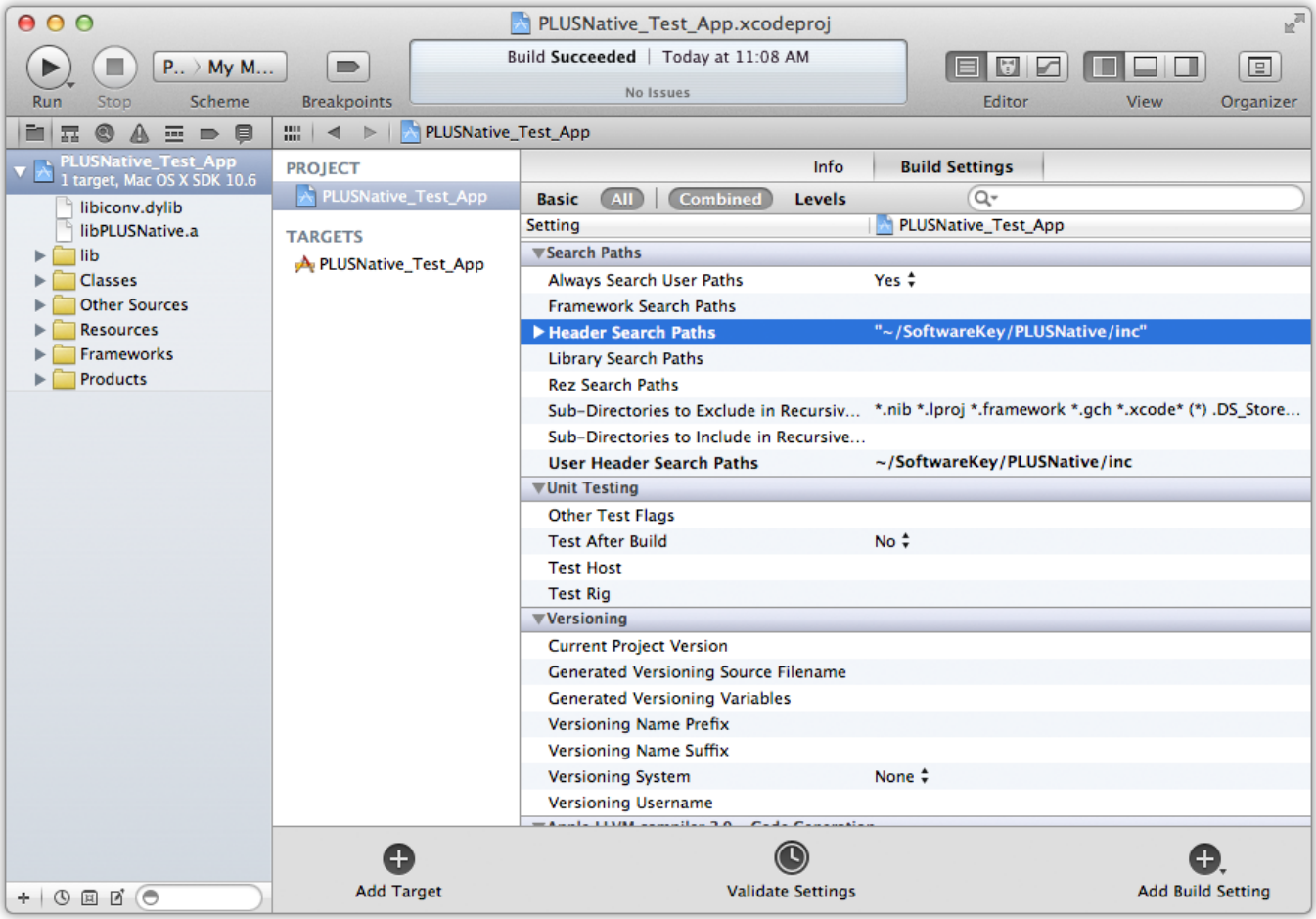
Debug Information

The PLUSNative libraries are not build with debug information. Consequently, if your linker options are configured to generate debug information, you may see linker warnings about not being able to locate PLUSNative.pdb. Under your project properties, you can select the Linker/Debugging section and set the "Generate Debug Info" property to "No" for your release build configuration(s). Alternatively, if you need to generate debug information for your program, you can go to the Linker/Command Line section and add the `/ignore:4099` switch to suppress this warning.

Adding PLUSNative to Xcode Project

Include Search Paths

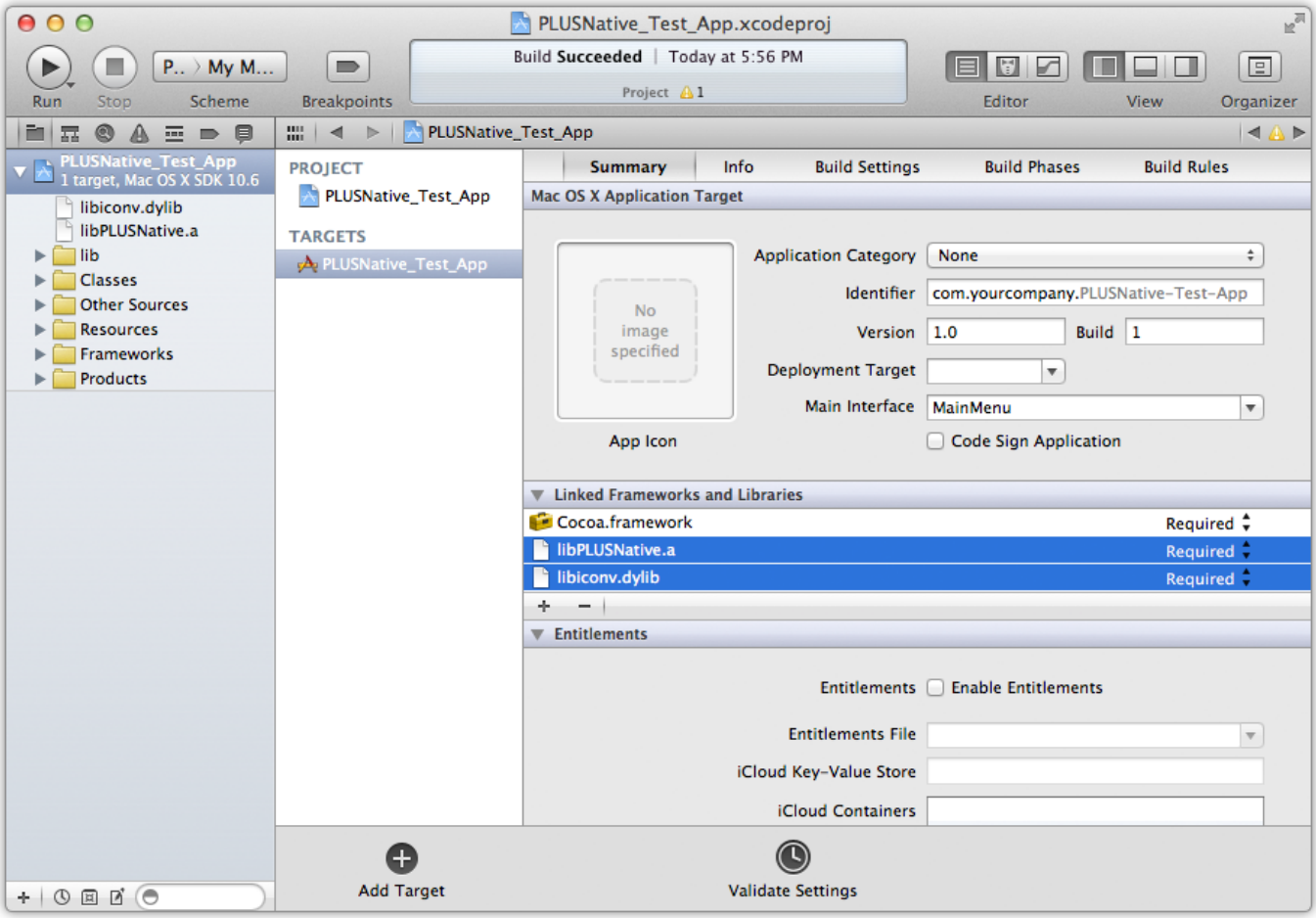
In order for the compiler to locate the *PLUSNative.h* header file that gets included with your source code you should add the *SoftwareKey/PLUSNative/inc* directory your Xcode project's include search path. Start by opening up your project's build settings. This is done by clicking the project name in the Project navigator and then clicking the project name under *PROJECT*. Click the **Build Settings** tab and select **All** to show all settings. Scroll down to the *Search Paths* section and double-click **Header Search Paths**. Click the plus (+) on the bottom left of the search paths dialog that pops up and add the path to the PLUSNative include directory (e.g. *SoftwareKey/PLUSNative/inc*).



Library Dependencies

You must configure your Xcode project to tell it which PLUSNative library to link with as well as any of it's **system library dependencies**. The libraries are added to your project's targets. Start by clicking the project name in the Project navigator and then clicking the target name under *TARGETS*, and then click the **Summary** tab. Click the plus (+) to the bottom left of the *Linked Frameworks and Libraries* section and add each of the PLUSNative library dependencies. Click the plus (+) again and click the **Add Other...** button and browse to the PLUSNative library directory and

choose the specific library you wish to link. In this example we browsed to *~/SoftwareKey/PLUSNative/lib/x64/macOS/static* and added *libPLUSNative.a*.



Adding PLUSNative to C++ Builder Project

The PLUSNative library for Windows is built with the Microsoft Visual C++ compiler, and does not ship with any import libraries for the Borland/Embarcadero C++ Builder compilers. Despite this, it is possible to use PLUSNative with these compilers with a few, trivial steps (after installing Protection PLUS 5 SDK):

Step 1: Generate an Import Library

The first step is to generate an import library, which will tell your C++ Builder application how to call the PLUSNative DLL. This can be done with a few easy commands using the Windows Command Prompt. (Note that in Windows Vista and higher, you may need to right-click on the "Command Prompt" shortcut in your start menu, and click the "Run as administrator" menu option to avoid "Access denied" errors.)

Windows Command Prompt

```
cd %PLUSNATIVE%\Library\Windows\DLL\x86
mkdir ..\..\CppBuilder
mkdir ..\..\CppBuilder\x86
implib ..\..\CppBuilder\x86\PLUSNative.lib PLUSNative.dll
```

Important

With old versions of C++ Builder, you may need to use the `-a` option when calling `implib` to avoid "Unresolved externals" errors from the linker.

Step 2: Configure your Project

With the import library generated, the next step is to configure your project.

- First, click the *Project/Options* menu, and then click *Directories/Conditionals*. Then add `;$(PLUSNATIVE)\Include` to the end of the *Include path* field, and add `;$(PLUSNATIVE)\Library\CppBuilder\x86` to the *Library path* field. (Be careful not to remove any of the contents originally in these fields - you should only be adding to the end of them.)
- Next, click the *Project/Add to project...* menu. Select "*Dynamic library import files (*.lib)*" (or "*Library file (*.lib)*" with older versions of C++ Builder) in the "*Files of type*" drop-down. Then open *%PLUSNATIVE%\Library\Windows\CppBuilder\x86\PLUSNative.lib* file.

Step 3: Start using PLUSNative

With the import library built and the project configured properly, you are now ready to use PLUSNative in your C++ Builder project. To do so, add an `#include "PLUSNative.h"` directive at the top of any relevant source file(s), but make sure this is above any `#pragma hdrstop` directives.

PLUSNative: Configuring the API Context

The PLUSNative library uses an API context to persist configuration settings and a copy of the license file in memory between consecutive function calls. Many of the functions in this library require an API context as the first argument. The API context is usually initialized during application start-up and must be disposed of properly after the license checks or on application shut-down.

Important

The steps laid out in the common implementation sub-topics of this manual do not match exactly what is in the **Protection PLUS 5 SDK sample projects**. The manual topics are more simplified to make it easier to explain, while the sample projects have more complex organization, but the resulting functionality is similar in both cases.

Initializing the API Context

During application start-up or prior to calling any of the functions in the PLUSNative library an API context must be initialized. The API context also determines whether to treat the **license file** as a read-only license file or a self-signed writable license file.

Important

- Once you configure which type of license file to use it should not be changed.
- If using both types of **license files** (read-only and writable) separate API contexts should be used for each type of license.
- Only a single license file may be used with an API context. If your application uses more than one license file, a separate API context should be used for each. (This does not apply to license file aliases, which are simply redundant copies of the license file.)

The following code snippet demonstrates initializing the API context:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;

//initialize API context (usually called during application start-up)
result = SK_ApiContextInitialize(SK_FLAGS_USE_SSL | SK_FLAGS_USE_ENCRYPTION | SK_FLAGS_USE_
SIGNATURE, FALSE,
                                 132400, 0, "1.0.0.0",
                                 "a9pbc8DqS6B8Kp8g5wkjF08s9h272bt8CsliasU8zavyvFsnFwCNEXZV7Qs0+KX...",
                                 &context);

//set the envelope key
result = SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, SK_APICONTEXT_ENVELOPEKEY,
"d+jLXiBpTMAqrunDrQb8kVGl+lCsZVg0zclbrQbXIX...");
```

Important

The example above uses data from the Instant SOLO Server test account. The Product ID, product version and encryption key data must be updated to use data specific to your own account. This is described in detail in the

Using your own SOLO Server account topic. The encryption envelope and envelope key above has been truncated for the sake of clarity.

Global Context Flags

When calling the `SK_ApiContextInitialize` function, the `flags` (first) argument accepts flags that can be applied as global context flags. In the example above, the `SK_FLAGS_USE_SSL`, `SK_FLAGS_USE_ENCRYPTION`, and `SK_FLAGS_USE_SIGNATURE` flags are specified globally for the new `context`. For any functions called using the `context` initialized in this function call, these flags are automatically applied in addition to any flags specified in the function call's `flags` argument. For instance, calling the `SK_CallXmlWebService` function with no flags (`SK_FLAGS_NONE`) with this `context` will automatically use the `SK_FLAGS_USE_SSL`, `SK_FLAGS_USE_ENCRYPTION`, and `SK_FLAGS_USE_SIGNATURE` flags as specified in this example. If this same function were to be called with `SK_FLAGS_REQUIRE_SSL`, then all 4 flags (`SK_FLAGS_USE_SSL`, `SK_FLAGS_USE_ENCRYPTION`, `SK_FLAGS_USE_SIGNATURE`, and `SK_FLAGS_REQUIRE_SSL`) would apply in this same example.

It is also possible to omit all global context flags in an individual function call using the `SK_FLAGS_EXPLICIT_ONLY` flag. To expand on the example above, let's say we wanted to make a web service call using SSL, but without separately encrypting and digitally signing the XML. (This could be done to call SOLO Server XML web service methods that do not support this extra layer of encryption, such as XmlLicenseServer's InfoCheck or UpdateCheck methods.) To do this with this example, this specific call to the `SK_CallXmlWebService` function would specify the `SK_FLAGS_EXPLICIT_ONLY` and `SK_FLAGS_USE_SSL` flags. The result would be that the call would *only* use the `SK_FLAGS_USE_SSL` flag, regardless of the flags that were previously defined when initializing the `context`.

Configuring Additional Options

Aside from the required configuration options set when initializing an API context, it is possible to set additional configuration options as well. One example is to set a **proxy server** configuration, as illustrated below:

C/C++

```
//set the proxy server
result = SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, SK_APICONTEXT_WEBSERVICE_PROXY,
"www.exampleproxy.com");

//set the username to use for proxy authentication
result = SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, SK_APICONTEXT_WEBSERVICE_
PROXYUSERNAME, "username");

//set the password to use for proxy authentication
result = SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, SK_APICONTEXT_WEBSERVICE_
PROXYPASSWORD, "password");
```

Disposing the API context

When done with the PLUSNative library, or on application shut-down, the API context must be disposed of properly to free memory that has been allocated during initialization and normal use. If you use multiple contexts and/or call `SK_ApiContextDispose()` multiple times, the last time you call the function you must pass in the **`SK_FLAGS_APICONTEXTDISPOSE_SHUTDOWN`** flag to shutdown the PLUSNative API and free all memory.

C/C++

```
//dispose of the API context (usually called during application shut-down)
result = SK_ApiContextDispose(SK_FLAGS_APICONTEXTDISPOSE_SHUTDOWN, &context);
```

PLUSNative: Calling SOLO Server XML Web Services

One can easily use the PLUSNative library to call any of the **SOLO Server web services** that are XML-based and accept string post data. For information on using the XML Activation Service Web Service please refer to the **Activating and getting a License File** topic in the manual. An overview of the available web services can be found in the SOLO Server manual's Web Services Overview topic.

As an example, this topic will demonstrate calling the XML License Service's `UpdateCheckS` web method, which can be used by your application to periodically check for new versions.

Building the Request

You must manually build the XML request document to pass to the web service. The XML helper functions provided by the PLUSNative library are recommended for this purpose. The SOLO Server manual documents the XML schema for each web methods request (the input) and response (the output) documents.

The following code snippet demonstrates building the request for the `UpdateCheckS` web method:

C/C++

```
//declare variables
char *url = "secure.softwarekey.com/solo/webservices/XmlLicenseService.asmx/UpdateCheckS";
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;
SK_XmlDoc request = NULL;
SK_XmlDoc response = NULL;
int resultCode = 0;
int statusCode = 0;
char *latestVersion = NULL;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions


//create a new XML document
result = SK_XmlDocumentCreateFromString(SK_FLAGS_NONE, "<LicenseUp-
dateCheck></LicenseUpdateCheck>", &request);

//append elements
result = SK_XmlElementAddNew(SK_FLAGS_NONE, request, NULL, "LicenseID", "123" );
result = SK_XmlElementAddNew(SK_FLAGS_NONE, request, NULL, "Password", "");
result = SK_XmlElementAddNew(SK_FLAGS_NONE, request, NULL, "ProductID", "0");
result = SK_XmlElementAddNew(SK_FLAGS_NONE, request, NULL, "ProductName", "My Product");
result = SK_XmlElementAddNew(SK_FLAGS_NONE, request, NULL, "LanguageCode", "en");
result = SK_XmlElementAddNew(SK_FLAGS_NONE, request, NULL, "VerifyDaysToDL", "False");
result = SK_XmlElementAddNew(SK_FLAGS_NONE, request, NULL, "Version", "1.0.0.0");
```

The above request is built using fictitious, hard-coded data for demonstration purposes. The code snippet above omits many of the parameters necessary to initialize the API context (the `context` variable). Refer to the **Configuring the API Context** topic for instructions and a complete example of calling the `SK_ApiContextInitialize` function.

Calling the Web Service

The code snippet below simply calls the web service using the request we built above. If the request contains a `PrivateData` node and the **API Context** has been configured to use the `SK_FLAGS_USE_ENCRYPTION` and `SK_FLAGS_USE_SIGNATURE` flags globally, the request will be encrypted and digitally signed with the call to `SK_CallXmlWebService`. (This does not apply to this example because it does not contain a `PrivateData` node in the request.)

C/C++

```
//call the web service
result = SK_CallXmlWebService(context, SK_FLAGS_NONE, url, request, &response, &resultCode,
&statusCode);

//free our request document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &request);

//check the result
if (SK_ERROR_NONE != result)
{
   //handle error condition
   ...
}
```

Determining the Result

When `SK_CallXmlWebService` returns `0` (zero), a valid response with no errors was received.

If you receive a result of `SK_ERROR_WEBSERVICE_RETURNED_FAILURE`, this indicates we successfully received a response, but SOLO Server encountered errors while processing the request. SOLO Server's web service error condition is returned in the `resultCode` parameter. References are available online for possible**SOLO Server result codes**. However, some SOLO Server web services have their own specific result codes that are documented in their respective web service topics in the manual. Many web services also return a human-readable error message that can be extracted from the response document's `ErrorMessage` node.

Parsing the Response

Once you determine the web service call succeeded, you must manually parse the response to determine what actions to take. Again, the XML helper functions are used here to extract the data out of the response document. The `SK_XmlNodeGetValue*` functions take an XPath to select the node in the document that you want to get the value of. In this example, the documents root node is `LicenseUpdateCheck` and we want to get the `LatestFreeVersion` node's value as a string. Normally, you would need to consider more than just this single node's value in a real-world example.

C/C++

```
//get the latest free version
result = SK_XmlNodeGetValueString(SK_FLAGS_NONE, response, "/Li-
censeUpdateCheck/LatestFreeVersion", FALSE, &latestVersion);

//free our response document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &response);

if (NULL != latestVersion)
{
```

```
   //display update notification to user
   ...
}

//free our string
SK_StringDispose(SK_FLAGS_NONE, &latestVersion);
```

Important

These basic source code examples omit important and necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it is important that you make sure each function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where a problem originated if and when one occurs.

PLUSNative: Activating and Getting a License File (Online Method)

Online activation is one option for activating an application and retrieving a license file from SOLO Server. Obtaining a license file from SOLO Server is required when working with read-only license files, since only SOLO Server has the encryption key data necessary to encrypt and digitally sign a read-only license file. Although it is possible to create a self-signed writable license file without using SOLO Server, it is often more convenient to obtain the base license file from SOLO Server that can then be modified as necessary. Either way, the procedure is largely the same. The only difference being how you configure the API context.

Important

Activation is only intended to occur once per system and/or user. DO NOT configure your application to activate automatically every time it is run, as this can cause many problems that affect the reliability of your application, and is considered a misuse of the SOLO Server activation web services. If you wish to validate the status of the license with SOLO Server in your application, use background checking. If you wish to offer a limited number of users an option to check-in/check-out seats for a license, consider using **Network Floating Licensing** features for this purpose.

The following example demonstrates activating an installation, retrieving a license file from SOLO Server, and then saving that license file to disk.

Generating the Request

The PLUSNative library has built-in functions for generating requests for the SOLO Server **XML Activation Service** and **XML License File Service** web services. Use of these functions avoids having to manually build the web service XML request document. If one finds it necessary, the XML request document can be modified using any of the XML helper functions. For instance, new elements can be added using the `SK_XmlElementAddNew` function.

The following code snippet demonstrates generating the activation request:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;
SK_XmlDoc request = NULL;
SK_XmlDoc response = NULL;
int resultCode = 0;
int statusCode = 0;
SK_XmlDoc license = NULL;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions

//generate the activation request
result = SK_SOLO_ActivateInstallationGetRequest(context, SK_FLAGS_NONE, 123, "password",
NULL, 1000, 1000, FALSE, "My Installation", NULL, &request, NULL);
```

The above request is generated using fictional hard-coded data for demonstration purposes. The code snippet above omits many of the parameters necessary to initialize the API context (the `context` variable). Refer to the **Configuring the API Context** topic for instructions and a complete example of calling the `SK_ApiContextInitialize` function.

Calling the Web Service

The code snippet below simply calls the web service using the request we built above. The request will be encrypted and digitally signed with the call to `SK_CallXmlWebService` provided the **API Context** has been configured to use the `SK_FLAGS_USE_ENCRYPTION` and `SK_FLAGS_USE_SIGNATURE` flags globally.

C/C++

```
//call the web service
result = SK_CallXmlWebService(context, SK_FLAGS_NONE, SK_CONST_WEBSERVICE_
ACTIVATEINSTALLATION_URL, request, &response, &resultCode, &statusCode);

//free our request document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &request);

//check the result
if (SK_ERROR_NONE != result)
{
   //handle error condition
   ...
}
```

The above call uses the `SK_CONST_WEBSERVICE_ACTIVATEINSTALLATION_URL` constant that's defined for use with Instant SOLO Server. If using a dedicated or externally-hosted SOLO Server, this will need updating to point to your specific domain.

Determining the Result

When `SK_CallXmlWebService` returns `0` (zero), a valid response with no errors was received.

If you receive a result of `SK_ERROR_WEBSERVICE_RETURNED_FAILURE`, this indicates we successfully received a response, but SOLO Server encountered errors while processing the request. SOLO Server's web service error condition is returned in the `resultCode` parameter. References are available online for possible**SOLO Server result codes**. Optionally, a human-readable error message that can be extracted from the response document's `ErrorMessage` node.

Parsing the Response

Once you determine the web service call succeeded, you must manually parse the response to determine what actions to take. Here, the XML helper functions are used here to extract the data out of the response document. A read-only license file is contained as a sub-document in the response's *'License'* node. Once you extract the license document from the response you can save it to disk.

C/C++

```
//get the license sub-document
result = SK_XmlNodeGetDocument(SK_FLAGS_NONE, response, "/Activ-
ateInstallationLicenseFile/PrivateData/License", &license);

//free our response document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &response);

if (NULL != license)
{
```

```
    //save the license file to disk
    result = SK_PLUS_LicenseFileSave(context, SK_FLAGS_NONE, filename, license);

    //free our license document
    SK_XmlDocumentDispose(SK_FLAGS_NONE, &license);
}
```

Although the above example saves the license file to disk, it can actually be saved anywhere, such as in a database. In such a scenario, you would only need the license string to save. If you don't save the license file using `SK_PLUS_LicenseFileSave`, it is necessary to call `SK_PLUS_LicenseLoad` to load the newly obtained license into memory.

Important

This basic example omits necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it's important you make sure the function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where the problem occurred.

PLUSNative: Manual Activation (Offline Method)

When Internet access is not available and online activation is not possible, manual activation is the recommend method for activating an installation and obtaining a license file from SOLO Server. This involves the end-user copying the activation request or saving it to a file that they would take to another computer with Internet access. They would then paste the request or upload the file to SOLO Server's manual request page, which processes the request and generates a response. The end-user must then copy this response or save it to a file and return to the computer they are attempting to activate. Finally, they must paste in the response or point the application to the file. The application can then parse the response and determine which action to take.

The following example demonstrates manually activating an installation, retrieving a license file from SOLO Server, and saving that license file to disk.

Generating and Encrypting the Request

The PLUSNative library has built-in functions for generating requests for the SOLO Server **XML Activation Service** and **XML License File Service** web services. Use of these functions avoids having to manually build the web service XML request document. If one finds it necessary, the XML request document can be modified using any of the XML helper functions. For instance, new elements can be added using the `SK_XmlElementAddNew` function. Once the request has been generated, it must be encrypted and digitally signed.

The following code snippet demonstrates generating the manual activation request:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;
SK_XmlDoc request = NULL;
SK_XmlDoc encryptedRequest = NULL;
char *encryptedRequestString = NULL;
int requestSize = 0;
SK_XmlDoc encryptedResponse = NULL;
SK_XmlDoc response = NULL;
int resultCode = 0;
char *originalSessionCode = NULL;
char *responseSessionCode = NULL;
SK_XmlDoc license = NULL;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions

//generate the activation request
result = SK_SOLO_ActivateInstallationGetRequest(context, SK_FLAGS_NONE, 123, "password",
NULL, 1000, 1000, FALSE, "My Installation", NULL, &request, &originalSessionCode);

//save the originalSessionCode for later validation
...

//encrypt/sign the request
result = SK_XmlDocumentEncryptRsa(context, SK_FLAGS_NONE, FALSE, &request,
&encryptedRequest);

//get our request as a string
result = SK_XmlDocumentGetDocumentString(SK_FLAGS_NONE, encryptedRequest, &
```

```
encryptedRequestString, &requestSize);

//free our encrypted request document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &encryptedRequest);

//allow the user to copy the request string or save it to a file
...
```

The above request is generated using fictional hard-coded data for demonstration purposes. The code snippet above omits many of the parameters necessary to initialize the API context (the `context` variable). Refer to the **Configuring the API Context** topic for instructions and a complete example of calling the `SK_ApiContextInitialize` function.

Using the SOLO Server Manual Request Page

Once the user generates the manual activation request, they must take it to a computer with Internet access where they can paste it in or upload it to the **SOLO Server manual request page**. If SOLO Server is able to successfully process the manual request it will generate a response that must be taken back to the computer being activated. Otherwise, it will display an error message and it may be necessary for the end-user to contact support for help and possibly have to start the process over.

Important

The above SOLO Server Manual Request link is for use with Instant SOLO Server only. If you are using a dedicated or externally-hosted SOLO Server, this will need to be updated to point to your specific domain.

Decrypting and Parsing the Response

After the end-user returns to the computer being activated with the encrypted response, it must be decrypted and verified. Once done, you must manually parse the response to determine what actions to take. Here, the XML helper functions are used here to extract the data out of the response document. A read-only license file is contained as a sub-document in the response's `License` node. Once you extract the license document from the response you can save it to disk.

C/C++

```
//allow the user to paste the response string or load it from a file
...

//load the response string into an XML document
result = SK_XmlDocumentCreateFromString(SK_FLAGS_NONE, responseString, &encryptedResponse);

//decrypt/verify the response
result = SK_XmlDocumentDecryptRsa(context, SK_FLAGS_NONE, FALSE, encryptedResponse,
&response);

//free our encrypted response document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &encryptedResponse);

//get the SessionCode from the response
result = SK_XmlNodeGetValueString(SK_FLAGS_NONE, response, "//SessionCode", FALSE, &respon-
seSessionCode);
```

```
//load our originalSessionCode that was previously saved

//verify the session codes match
if ((NULL != responseSessionCode) || (NULL == strstr(originalSessionCode,
responseSessionCode)))
{
   //handle error condition
   ...
}

//clear our originalSessionCode that was previously saved
...

//free our session code strings
SK_StringDispose(SK_FLAGS_NONE, &originalSessionCode);
SK_StringDispose(SK_FLAGS_NONE, &responseSessionCode);

//get the license sub-document
result = SK_XmlNodeGetDocument(SK_FLAGS_NONE, response, "/Activ-
ateInstallationLicenseFile/PrivateData/License", &license);

//free our response document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &response);

if (NULL != license)
{
   //save the license file to disk
   result = SK_PLUS_LicenseFileSave(context, SK_FLAGS_NONE, filename, license);

   //free our license document
   SK_XmlDocumentDispose(SK_FLAGS_NONE, &license);
}
```

Although the above example saves the license file to disk, it can actually be saved anywhere, such as in a database. In such a scenario, your application would only need to save the `license` string. If you don't save the license file using **SK_PLUS_LicenseFileSave**, it is necessary to call **SK_PLUS_LicenseLoad** to load the newly obtained license into memory.

Important

If the user is pasting in a response instead of loading a response file, there are a few potential issues to be aware of:

- Users may be limited by the size of their clipboard if the response is too large. An"Invalid Data" error will occur if the entire response is not pasted in.
- For macOS users, if the encrypted response contains the characters "TM" and the user pastes in the response as a string, the operating system will automatically convert the "TM" characters to the trademark superscript character, which invalidates the response.

These issues do not occur when the encryption response is loaded from a file.

Session Code Validation

This example uses session code validation to prevent the end-user from processing the same response multiple times, which is known as a replay attack. A random session code is generated and included in each web service request. This session code must be saved somewhere on the system so it can later be retrieved and compared to the session code found in the web service response. If the session codes match then you know the response was the response associated with the original web service request. Otherwise, the response should be consider invalid.

Important

These basic source code examples omit important and necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it is important that you make sure each function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where a problem originated if and when one occurs.

Using Trigger Codes for Telephone Activations

Trigger code validation is a means of achieving software activation through the manual exchange of numeric values (in other words, it is a challenge-response mechanism). Review the **overview on trigger codes** for additional, high-level details on how this works and how to generate activation codes.

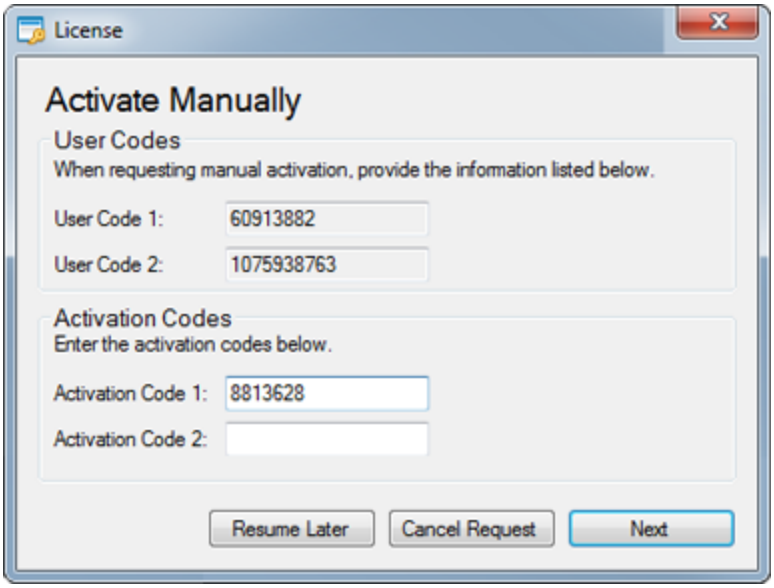
Important

Keep in mind that any challenge-response mechanisms such as trigger code validation can be reverse engineered (similar to how "key generators" are available on the Internet for many popular software applications). Therefore, it is best to use Protection PLUS 5 SDK's standard **online** and **manual** activation features when possible, which offer much stronger **security** than trigger codes.

Adding Support for Trigger Codes to your Application

Before you can add support for any form of activation, your application must first **initialize the context**. Once activation has been implemented, the application will also need to include **copy protection** to ensure it only runs on devices in which it was activated.

To process trigger code activations, you will need to collect input from the user using a form/dialog or some other means. Most activation forms/dialogs are pretty similar, and can be designed similar to the illustration below.



As shown above, your application would need to show the User Code 1 and User Code 2 values to the user. The example code below outlines how you can add support for trigger code activations to your application.

C/C++

```
#define TRIGGER_CODE_SEED (400)
#define REGKEY2_SEED  (123)

int userCode1 = 0;
int userCode2 = 0;
int activationCode1 = 0;
```

```
int activationCode2 = 0;
int triggerCodeNumber = 0;
int eventData = 0;
SK_ResultCode res = SK_ERROR_NONE;

res = SK_PLUS4_GenerateUserCode1Value(SK_FLAGS_NONE, &userCode1);
if (SK_ERROR_NONE != res)
{
   //TODO: The User Code 1 value could not be generated. Add error handling here!
}
else
{
   //Show the end-user the value generated and stored in userCode1.
   //When adding support for delayed trigger codes, here is where you would store the user-
   Code1 value for later use.
}

res = SK_PLUS4_GenerateUserCode2Value(context, SK_FLAGS_NONE, &userCode2);
if (SK_ERROR_NONE != res)
{
   //TODO: The User Code 2 value could not be generated. Add error handling here!
}
else
{
   //Show the end-user the value generated and stored in userCode2.
}

//Prompt the user to enter the activation codes here, and store these values in activ-
ationCode1 and activationCode2.

res = SK_PLUS4_ValidateTriggerCode(SK_FLAGS_NONE, activationCode1, activationCode2,
userCode1, userCode2, TRIGGER_CODE_SEED, REGKEY2_SEED, &triggerCodeNumber, &eventData);
//When adding support for delayed trigger codes, here is where you would clear the stored the
userCode1 value.
if (SK_ERROR_NONE != res)
{
   //TODO: The activation codes failed validation. Add error handling here.
}
else
{
   //The activation codes were validated successfully.
   //TODO: Add your trigger code processing here! A rough example is outlined below.
   switch (triggerCodeNumber)
   {
      case 1:
         //As an example, code could be added here so that trigger code 1 activates your
         application as a full, non-expiring (or perpetual) license.
         break;
      case 10:
         //As an example, code could be added here so that trigger code 10 activates your
         application as a time-limited (or lease/periodic) license.
         //You could use the value in eventData to determine the number of days until the
         license expires by setting your license's EffectiveEndDate property using a value
```

```
        derived from SK_DateTimeGetCurrentString and SK_DateTimeAddDays.
        break;
     default:
        //TODO: An unsupported trigger code number was specified. Add error handling here.
        break;
   }
}
```

Delayed Trigger Codes

A *delayed trigger* code is simply where you allow the user to process a trigger code activation after having closed and re-launched your application. The problem this solves primarily revolves around how the activation form/dialog initializes the User Code 1 value, as this value is randomized. Although this is not typically an issue with telephone activations, this can pose a challenge when the user might email, fax, or text the user code values, close the application, and re-launch the application to complete the activation later. If your application needs to support this kind of delay between when the user sends the user code values and when the user completes the activation, your application may need to implement this feature by storing the user code 1 value somewhere. This can be stored in a hidden file or registry key of your choosing, and it is strongly recommended that you consider encrypting the value as well.

PLUSNative: Background Checking and Refreshing Licenses

Checking the status of an activated license by **calling SOLO Server web services** is an effective and efficient way to check if activated licenses remain authorized. These checks can prevent copies of your application from running in scenarios like the following.

- A user purchases a license for your application and activates it, but later requests refund. When processing the refund, the license status should be changed in SOLO Server to indicate the protected application is no longer authorized to run.
- Similar to above, a user purchases your application, but does so with fraudulent payment information, which could result in a charge-back. If the charge-back cannot be resolved in your favor, the license status should be changed in SOLO Server to indicate the protected application is no longer authorized to run.
- A user installs and activates on one computer, some time passes, and that computer experiences some kind of failure that renders the prior activation unusable according to the user. You may then log into SOLO Server, deactivate that computer's "Installation ID" so that specific computer is no longer authorized, and the user may then be allowed to activate on a new computer.

In each case, the background check will verify the system on which the application was activate is still authorized by verifying the status of the license and the individual system's activation are still valid. If the background check indicates the license is no longer authorized for the activated system, then your application can take steps needed to disable itself as needed.

Important

Be mindful about whether or not disabling access to your application is appropriate. For example, if your application requires high availability (such as a web application or service) and/or your application serves some mission-critical function, it may be more appropriate to have your application display nag messages and/or disable only non-critical features so as to avoid disrupting your customers' business or operation.

Important

When using background checking or license refreshing features, it is very important that you test this functionality in your application while running in an environment where connections are refused or unavailable, and also where connections time out. Doing so can help you identify areas of your application that may become unresponsive or unstable when your application encounters these conditions.

Background Checking

The XML Activation Service web service has a `CheckInstallationStatus` method, which your application can leverage to check the status of the license and the status of the system's activation. This is particularly ideal when your application needs to check its status frequently (such as every time the application runs, or every time a critical feature of your application is used if it is always running).

Generating the Request

The following code snippet demonstrates generating the activation request:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;
SK_XmlDoc request = NULL;
SK_XmlDoc response = NULL;
```

```
int resultCode = 0;
int statusCode = 0;
SK_XmlDoc license = NULL;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions

//generate the activation request
result = SK_SOLO_CheckInstallationStatusGetRequest(context, SK_FLAGS_NONE, "UB6AL-MAVGB-
A2YWX-2Q4QL-UCKMB-2", &request, NULL);
```

The above request is generated using fictional, hard-coded Installation ID for demonstration purposes. The code snippet above omits many of the parameters necessary to initialize the API context (the `context` variable). Refer to the **Configuring the API Context** topic for instructions and a complete example of calling the `SK_ApiContextInitialize` function.

Calling the Web Service

The code snippet below simply calls the web service using the request we built above. The request will be encrypted and digitally signed with the call to `SK_CallXmlWebService` provided the **API Context** has been configured to use the `SK_FLAGS_USE_ENCRYPTION` and `SK_FLAGS_USE_SIGNATURE` flags globally.

C/C++

```
//call the web service
result = SK_CallXmlWebService(context, SK_FLAGS_NONE, SK_CONST_WEBSERVICE_CHECKINSTALLATION_
URL, request, &response, &resultCode, &statusCode);

//free our request document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &request);

//check the result
if (SK_ERROR_NONE != result)
{
   //handle error condition
   ...
}
```

The above call uses the `SK_CONST_WEBSERVICE_CHECKINSTALLATION_URL` constant that is defined for use with Instant SOLO Server. If you are using a dedicated or externally-hosted SOLO Server, define and use a new constant in your application's source code which points to your specific domain.

Determining the Result

When `SK_CallXmlWebService` returns `0` (zero), a valid response with no errors was received.

If you receive a result of `SK_ERROR_WEBSERVICE_RETURNED_FAILURE`, this indicates we successfully received a response, but SOLO Server encountered errors while processing the request. SOLO Server's web service error condition is returned in the `resultCode` parameter. References are available online for possible **SOLO Server result codes**. Optionally, a human-readable error message that can be extracted from the response document's `ErrorMessage` node.

Important

This basic example omits necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it's important you make sure the function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where the problem occurred.

Refreshing Licenses

The XML License File Service web service has a `GetLicenseFile` method, which your application can leverage to check the status of the license and the status of the system's activation while simultaneously retrieving an updated copy of the license file. This is particularly ideal when your application needs to get updated license information. Since the license contains details about when the license expires, the customer who owns the license, custom data you specified, and much more, this is an easy way to get a complete update to pull down any and all changes all in a single web service call.

Refreshing the license file requires much more server and application resources, so it is important avoid using this feature too frequently in your protected applications. The **license file** contains a `SignatureDate` property, which tells you when the license file was last issued by SOLO Server. This field can be leveraged to determine the last time the license was validated and refreshed with SOLO Server, and allows you to implement logic to only attempt and/or require your application to phone-home after a certain amount of time has passed. All of the Protection PLUS 5 SDK samples include settings or properties which you may reference and re-configure as needed for your application.

Generating the Request

The following code snippet demonstrates generating the activation request:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;
SK_XmlDoc request = NULL;
SK_XmlDoc response = NULL;
int resultCode = 0;
int statusCode = 0;
SK_XmlDoc license = NULL;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions

//generate the activation request
result = SK_SOLO_GetLicenseFileGetRequest(context, SK_FLAGS_NONE, "UB6AL-MAVGB-A2YWX-2Q4QL-
UCKMB-2", &request, NULL);
```

The above request is generated using fictional, hard-coded Installation ID for demonstration purposes. The code snippet above omits many of the parameters necessary to initialize the API context (the `context` variable). Refer to the **Configuring the API Context** topic for instructions and a complete example of calling the `SK_ApiContextInitialize` function.

Calling the Web Service

The code snippet below simply calls the web service using the request we built above. The request will be encrypted and digitally signed with the call to `SK_CallXmlWebService` provided the **API Context** has been configured to use the `SK_FLAGS_USE_ENCRYPTION` and `SK_FLAGS_USE_SIGNATURE` flags globally.

C/C++

```
//call the web service
result = SK_CallXmlWebService(context, SK_FLAGS_NONE, SK_CONST_WEBSERVICE_GETLICENSEFILE_URL,
request, &response, &resultCode, &statusCode);

//free our request document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &request);

//check the result
if (SK_ERROR_NONE != result)
{
   //handle error condition
   ...
}
```

The above call uses the `SK_CONST_WEBSERVICE_GETLICENSEFILE_URL` constant that is defined for use with Instant SOLO Server. If you are using a dedicated or externally-hosted SOLO Server, define and use a new constant in your application's source code which points to your specific domain.

Determining the Result

When `SK_CallXmlWebService` returns `0` (zero), a valid response with no errors was received.

If you receive a result of `SK_ERROR_WEBSERVICE_RETURNED_FAILURE`, this indicates we successfully received a response, but SOLO Server encountered errors while processing the request. SOLO Server's web service error condition is returned in the `resultCode` parameter. References are available online for possible **SOLO Server result codes**. Optionally, a human-readable error message that can be extracted from the response document's `ErrorMessage` node.

Important

This basic example omits necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it's important you make sure the function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where the problem occurred.

Revoking or Removing a License

When determining the result of a web service call made to either `CheckInstallationStatus` or `GetLicenseFile`, you may decide that certain responses received should revoke or remove your application's license. For example, you might take this sort of action when you receive a result code of 5010 (invalid Product ID), 5015 (invalid Installation ID), 5016 (deactivated Installation ID), or 5017 (invalid license status). Deleting the license file is appropriate when using read-only **license files**. However, if you use writable license files, you will need to add logic to update the license file to correctly reflect the new state of the license (i.e. you could make it act like an expired evaluation/trial).

Important

Be mindful about whether or not disabling access to your application is appropriate. For example, if your application requires high availability (such as a web application or service) and/or your application serves some mission-critical function, it may be more appropriate to have your application display nag messages and/or disable only non-critical features so as to avoid disrupting your customers' business or operation.

PLUSNative: Deactivating and Transferring Licenses

Deactivation can occur one of two ways: you may allow customers to deactivate from your application, or you can remotely **deactivate a license or an activation using SOLO Server's administrative interface**.

Important

It is very important to rely on background checking and refreshing when allowing users to deactivate your application! Automated background checking helps prevent users from licensing more systems than intended by restoring the entire system/computer from an image or snapshot taken before deactivation occurred.

In either case, when your application deactivates its license, or runs a **background check or refreshes its license** and detects that it has been deactivated, it can then disable access to the application or certain features.

Important

Be mindful about whether or not disabling access to your application is appropriate. For example, if your application requires high availability (such as a web application or service) and/or your application serves some mission-critical function, it may be more appropriate to have your application display nag messages and/or disable only non-critical features so as to avoid disrupting your customers' business or operation.

Deactivating and Transferring Licenses in your Application

Allowing your customers to do deactivate individual activations online is a convenience to them that also reduces your support overhead. Since deactivating allows users to **activate** a new system, this is a very simple and convenient way for your customers to essentially migrate (or transfer) the license from one system to another. An example of when a customer would want to use this functionality would be when he or she is upgrading from one computer to another.

Generating the Request

The following code snippet demonstrates generating the activation request:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;
SK_XmlDoc request = NULL;
SK_XmlDoc response = NULL;
int resultCode = 0;
int statusCode = 0;
SK_XmlDoc license = NULL;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions

//generate the activation request
result = SK_SOLO_DeactivateInstallationGetRequest(context, SK_FLAGS_NONE, "UB6AL-MAVGB-A2YWX-
2Q4QL-UCKMB-2", &request, NULL);
```

The above request is generated using fictional, hard-coded Installation ID for demonstration purposes. The code snippet above omits many of the parameters necessary to initialize the API context (the `context` variable). Refer to the

Configuring the API Context topic for instructions and a complete example of calling the `SK_ApiContextInitialize` function.

Calling the Web Service

The code snippet below simply calls the web service using the request we built above. The request will be encrypted and digitally signed with the call to `SK_CallXmlWebService` provided the **API Context** has been configured to use the `SK_FLAGS_USE_ENCRYPTION` and `SK_FLAGS_USE_SIGNATURE` flags globally.

C/C++

```
//call the web service
result = SK_CallXmlWebService(context, SK_FLAGS_NONE, SK_CONST_WEBSERVICE_
DEACTIVATEINSTALLATION_URL, request, &response, &resultCode, &statusCode);

//free our request document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &request);

//check the result
if (SK_ERROR_NONE != result)
{
   //handle error condition
   ...
 }
```

The above call uses the `SK_CONST_WEBSERVICE_DEACTIVATEINSTALLATION_URL` constant that is defined for use with Instant SOLO Server. If you are using a dedicated or externally-hosted SOLO Server, define and use a new constant in your application's source code which points to your specific domain.

Determining the Result

When `SK_CallXmlWebService` returns `0` (zero), a valid response with no errors was received.

If you receive a result of `SK_ERROR_WEBSERVICE_RETURNED_FAILURE`, this indicates we successfully received a response, but SOLO Server encountered errors while processing the request. SOLO Server's web service error condition is returned in the `resultCode` parameter. References are available online for possible **SOLO Server result codes**. Optionally, a human-readable error message that can be extracted from the response document's `ErrorMessage` node.

Important

This basic example omits necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it's important you make sure the function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where the problem occurred.

Revoking or Removing the License

Once you have determined the result of `DeactivateInstallation` indicates a successful deactivation with SOLO Server, you may then have your application revoke or remove its license. Deleting the license file is appropriate when using read-only **license files**. However, if you use writable license files, you will need to add logic to update the license file to correctly reflect the new state of the license (i.e. you could make it act like an expired evaluation/trial).

PLUSNative: Reading License Fields

Once you activate and get a license file from SOLO Server, you must then determine if the license is valid . When an application starts up, it attempts to load it's license file and read various fields to determine the state of the license. Depending on the licensing requirements, you may need to determine if the license is expired or which features to enable based on the value of certain user-defined fields. Aside from ensuring the license is valid, you may also want to display registered user information, such as on the About screen.

The following example demonstrates reading the `InstallationID` field from the license file.

Loading a License File

The following code snippet loads a license file from disk:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;
char *filename = ...;
SK_XmlDoc license = NULL;
char *installationId = NULL;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions

//load the license file
result = SK_PLUS_LicenseFileLoad(context, SK_FLAGS_NONE, filename);
```

Since there are numerous locations where the license file could be saved, and since acceptable locations vary from platform to platform, the code snippet above omits the actual path to the license file. This code snippet also omits many of the parameters necessary to initialize the API context (the *context* variable). Refer to the **Configuring the API Context** topic for instructions and a complete example of calling the `SK_ApiContextInitialize` function.

Loading a License File from an Image

Important

The steganography feature is presently a beta, and changes may be applied which could require minor source code modifications to be made in order to use a future release. Please keep in mind that **your testing** should be very thorough. Also, use your own images distributed with your application such as a splash screen image. Do not use an existing image such as the desktop wallpaper since another product could share that license location.

If the license file path is a **supported image format**, **steganography** is automatically used and the license information is read from the image file.

Reading License Fields

After the license is loaded, it is stored in memory in the API context. You can get a copy of the decrypted license document and read the various fields using the XML helper functions. Depending on the field being accessed, there are functions for parsing the field and returning the value as a string, an integer, a floating-point integer or as a date.

Below we read the `InstallationID` field as a string value:

C/C++

```
//get the decrypted license document
result = SK_PLUS_LicenseGetXmlDocument(context, SK_FLAGS_NONE, &license);

//read the InstallationID field
result = SK_XmlNodeGetValueString(SK_FLAGS_NONE, license, "/Soft-
wareKey/PrivateData/License/InstallationID", FALSE, &installationId);

if (NULL != installationId)
{
   //use the installationId string, such as in a web service call
   ...

   //free our installationId string
   SK_StringDispose(SK_FLAGS_NONE, &installationId);
}

//free our license document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &license);
```

Important

These basic source code examples omit important and necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it is important that you make sure each function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where a problem originated if and when one occurs.

PLUSNative: Using Self-Signed Writable License Files

Using a read-only license file issued by SOLO Server is the recommended approach since it is inherently more secure. However, it may be desirable to use a self-signed writable license file for a variety of reasons (**read more about license files**). This is often the case when implementing unmanaged evaluations/trials that do not require activation, or if the application must keep track of usage data while offline. When using writable license files it is recommended to use redundant copies of the license file, known as license file aliases. These are used to help prevent an end-user from backing up and restoring a license file or from reinstalling an evaluation to gain additional unauthorized usage. The last date and time the application was ran is generally stored in the `LastUpdated` field in the license file. When the license file and aliases are loaded their `LastUpdated` date/times are compared to determine which one is newer, and older copies will automatically be overwritten.

The following example demonstrates loading a license file and it's aliases and updating the `LastUpdated` date/time with the current system time.

Loading a License File and License File Aliases

The license file aliases are automatically loaded when the main license file is loaded. If the main license file has been backed up and restored, it will be automatically overwritten by the newer aliases. If the main license file has been deleted, it will be restored from an aliases file if present.

Important

The steganography feature is presently a beta, and changes may be applied which could require minor source code modifications to be made in order to use a future release. Please keep in mind that **your testing** should be very thorough. Also, use your own images distributed with your application such as a splash screen image. Do not use an existing image such as the desktop wallpaper since another product could share that license location.

If the license file or alias files are a **supported image file** then **steganography** will automatically be used to read and write the license to the specified image.

The following code snippet loads a license file and several license file aliases from disk:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;
char *filename = ...;
char *alias1 = ...;
char *alias2 = ...;
SK_XmlDoc license = NULL;
char *lastUpdated = NULL;
char *now = NULL;
int comparison = 0;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions

//define our license file aliases
result = SK_PLUS_LicenseAliasAdd(context, SK_FLAGS_NONE, alias1);
result = SK_PLUS_LicenseAliasAdd(context, SK_FLAGS_NONE, alias2);

//load the license file (and aliases)
result = SK_PLUS_LicenseFileLoad(context, SK_FLAGS_NONE, filename);
```

Since there are numerous locations where the license file could be saved, and since acceptable locations vary from platform to platform, the code snippet above omits the actual path to the license file and license file aliases. This code snippet also omits many of the parameters necessary to initialize the API context (the `context` variable). Refer to the **Configuring the API Context** topic for instructions and a complete example of calling the `SK_ApiContextInitialize` function.

Configuring Aliases

The first step is to add some code to your license implementation class constructor. The code added will vary depending on what locations you use for your aliases.

Important

The example code below and in any sample application code shows locations that are only meant to provide some guidance on selecting your application's alias locations. Your application must use different alias names and/or locations than any of the examples given by this documentation and any of the sample applications.

Additionally, it is equally important that any two applications (which do not share a license) use separate license file and alias locations.

User-Specific Locations

User-specific alias and license file locations are convenient because they can help you to avoid problems that need to be addressed when users with limited access to a system try to use your software. More specifically, if your application were to use user-specific locations exclusively, the primary benefit would be that users would never need elevated access to run and activate your application. However, the significant drawback is that because the locations used are unique to each user which signs-on to a system, each user will have his or her own license file and aliases, meaning each user will need to activate to be able to use the application. Some examples of user-specific alias locations are described below.

- In a user-specific data folder, such as the user's AppData folder (in Windows), or another location under the user's home directory.
- In a user-specific registry location (Windows only), such as Software\[your product's name] under the HKEY_CURRENT_USER hive, or Software\Classes\[randomly generated GUID/UUID]\Programmable:Version under the HKEY_CURRENT_USER hive. (You can use SK_SessionCodeGenerate to generate a random GUID/UUID.)

Global Locations

Important

Microsoft Office applications (such as Word, Excel, Access, etc...) may run in a **ClickToRun** environment. This environment has known limitations that make it problematic for licensed Office add-ins and macros to use global locations. Consequently, licensed add-ins and macros that target these environments should only use user-specific locations for licenses and aliases. See **this knowledge-base article** for more details.

Global alias and license file locations are locations which are shared by all users on a system. The advantage to using global locations is that the application only needs to be activated once on a given system. However, the drawback to this is that you typically need to consider setting permissions on these locations when **deploying your application**. Some examples for global alias locations are provided below.

- In Windows, you may opt to use
  - If the protected application is installed in a global location (e.g. in the C:\Program Files folder in Windows), you may be able to store the aliases in the same location as the application's executable file.
  - A common application data folder, such as C:\Program Data, the Public Documents folder, or another known/common **folder location**.

    - In a system folder, such as C:\Windows\System32.
    - In a global registry area, such as clsid under the HKEY_CLASSES_ROOT hive, or another area under Software under the HKEY_LOCAL_MACHINE hive.
- In macOS, you may opt to
    - Include an alias somewhere in your application bundle folder.
    - Alternatively, you can also consider the /Application Support folder and other folders documented in the **Mac developer documentation**.
- In Linux, you can consider using the following depending on the nature of your application:
    - Files located under an application-specific directory structure under the /opt directory.
    - A location that adheres to the Filesystem Hierarchy Standard (**FHS**).
    - A shared resource location (e.g. your own custom sub-directory structure under /usr/shared), assuming a package is used to simplify installation and removal of your application.

Writing License Fields

After making any changes to the license file the `LastUpdated` date/time field should be updated prior to saving. When saving the license file all license file aliases are automatically updated as well.

The following code snippet demonstrates reading the `LastUpdated` date/time field and updating it with the current date/time, provided the `LastUpdated` date/time is in the past. This helps prevent the user from backdating the system clock.

C/C++

```
//get the decrypted license document
result = SK_PLUS_LicenseGetXmlDocument(context, SK_FLAGS_NONE, &license);

//read the LastUpdated date/time field
result = SK_XmlNodeGetValueDateTimeString(SK_FLAGS_NONE, license, "/Soft-
wareKey/PrivateData/License/LastUpdated", &lastUpdated);

//get the current system date/time
result = SK_DateTimeGetCurrentString(SK_FLAGS_NONE, &now);

//make sure the LastUpdated date/time is in the past or the current system time.
result = SK_DateTimeCompareStrings(SK_FLAGS_NONE, lastUpdated, now, &comparison);
if (comparison <= 0)
{
   //update the LastUpdated date/time field
   result = SK_XmlNodeSetValueDateTimeString(SK_FLAGS_NONE, license, "/Soft-
   wareKey/PrivateData/License/LastUpdated", now);

   //save our changes
   result = SK_PLUS_LicenseFileSave(context, SK_FLAGS_NONE, filename, license);
}

//free our license document
SK_XmlDocumentDispose(SK_FLAGS_NONE, &license);

//free our date-time strings
SK_StringDispose(SK_FLAGS_NONE, &lastUpdated);
SK_StringDispose(SK_FLAGS_NONE, &now);
```

Important

These basic source code examples omit important and necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it is important that you make sure each function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where a problem originated if and when one occurs.

PLUSNative: Adding Basic Copy Protection

A major aspect of software licensing is copy protection. This prevents an end-user from activating a license on one system and copying that application to another system to gain additional unauthorized licenses. A machine fingerprint is generated and saved in each license file. This fingerprint consists of several attributes known as system identifiers, which may consist of the network adapter's MAC address, the computer name, or other uniquely identifiable data. Each time an application runs, it generates the fingerprint for the current system and compares it to that stored in the license file. If enough of the individual system identifiers match, it is reasonable to believe it is the same system and the application is authorized to run. Otherwise, the user must reactivate. The system fingerprint is also included in the activation request and sent to SOLO Server. If the product option is configured to *"Allow Reactivations on Same Computer"* then SOLO Server will look for a previously activated installation with a matching fingerprint, and if found, will allow the system to reactivate.

The following example demonstrates generating the system fingerprint and comparing that to the fingerprint in the license file to determine if the application is authorized to run.

Initializing System Identifiers

There are several built-in system identifier algorithms that can be used to fingerprint a system. The fingerprint is usually only generated on application start-up and gets stored in memory in the API context.

Note

More detailed information on each system identifier can be found in the PLUSNative API documentation under the **SK_SystemIdentifierAlgorithm Enumerations**.

The following code snippet shows how to initialize the current system's identifiers:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
SK_ApiContext context = NULL;
int count = 0;
int matches = 0;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions

//generate the current system identifiers
result = SK_PLUS_SystemIdentifierAlgorithmAddCurrentIdentifiers(context, SK_FLAGS_NONE, SK_
SYSTEM_IDENTIFIER_ALGORITHM_NIC, NULL, &count);
result = SK_PLUS_SystemIdentifierAlgorithmAddCurrentIdentifiers(context, SK_FLAGS_NONE, SK_
SYSTEM_IDENTIFIER_ALGORITHM_COMPUTER_NAME, NULL, &count);
result = SK_PLUS_SystemIdentifierAlgorithmAddCurrentIdentifiers(context, SK_FLAGS_NONE, SK_
SYSTEM_IDENTIFIER_ALGORITHM_HARD_DISK_VOLUME_SERIAL, NULL, &count);
```

The code snippet above omits many of the parameters necessary to initialize the API context (the *context* variable). Refer to the **Configuring the API Context** topic for instructions and a complete example of calling the `SK_ApiContextInitialize` function.

Basic System Identifier Validation

Once the license file has been loaded, the current system identifiers can be compared to that in the license file.

C/C++

```
//compare the current system identifiers to that in the license file
result = SK_PLUS_SystemIdentifierCompare(context, SK_FLAGS_NONE, NULL, &count, &matches);

//make sure all system identifiers match
if (matches < count)
{
   //the current system is not authorized
   ...
 }
```

The code above checks to make sure the system's current identifiers are an exact match to the identifiers authorized during activation. If the CurrentIdentifiers do not match the AuthorizedIdentifiers in the license file exactly, then the license is rejected by the example above.

The `SK_PLUS_SystemIdentifierCompare` function's `type` argument can be used to filter out the comparison to a certain system identifier algorithm type. This can be useful if you want to implement a more granular **system identification** and matching algorithm.

Comparing System Identifiers with Fuzzy-Matching

It is possible to customize your validation logic to allow for some amount of acceptable change/variation by comparing the hash values of certain types of `CurrentIdentifiers` against the hash values of the same type of `AuthorizedIdentifiers`.

C/C++

```
//compare the current HardDiskVolumeSerialIdentifiers to that in the license file
result = SK_PLUS_SystemIdentifierCompare(context, SK_FLAGS_NONE, "HardDiskVolumeSeri-
alIdentifier", &count, &matches);

//make sure at least one system identifier matches
if (matches < 1)
{
   //the current system is not authorized
   ...
}
```

The `SK_PLUS_SystemIdentifierCompare` function's `type` argument can be used to filter out the comparison to a certain system identifier algorithm type. This can be useful if you want to implement a more granular **system identification** and matching algorithm.

Custom System Identifier Algorithms

Depending on the licensing requirements, the built-in system identifier algorithms might not be suitable. For example, you may wish to bind the license to a particular user by implementing a username identifier algorithm.

The following code snippet shows how to initialize a custom system identifier:

C/C++

```
//declare variables
SK_ResultCode result = SK_ERROR_NONE;
```

```
SK_ApiContext context = NULL;
char *value = ...;

//initialize API context (usually called during application start-up)
//refer to the Configuring the API Context topic for instructions

//generate the current system identifiers
result = SK_PLUS_SystemIdentifierAddCurrentIdentifier(context, SK_FLAGS_NONE, "Cus-
tomIdentifier1", "CustomIdentifier", value);
```

The `value` variable in the above code snippet would be initialized to a plain-text string value consisting of the custom system identifier data. The **`SK_PLUS_SystemIdentifierAddCurrentIdentifier`** function will hash the identifier data so that no user-identifiable data is transmitted or stored to avoid privacy concerns.

Important

These basic source code examples omit important and necessary error checking for the sake of clarity. Many of the functions used could fail for one reason or another, and it is important that you make sure each function call succeeds before passing the result from one function to another. Otherwise, you may not be able to tell exactly where a problem originated if and when one occurs.

PLUSNative: Validating Time-Limited and Subscription Licenses

Validating time-limited licenses is simple in principle. To do this you can simply check to make sure the system's current date is not past the license file's `EffectiveEndDate`. However, the tricky part is checking to make sure your application can trust the licensed system is reporting a reasonably accurate date. PLUSNative offers a variety of ways to validate the system's date, and it is up to you to pick and chose the methods most appropriate and reliable for your application based upon its needs and the expectations of the environments in which the application will run. If your licensing requirements never impose any time limits, then you may not need to worry about validating the licensed system's clock (although it would still be good to read through and understand the subject for future reference).

Date and Time Limitations

Important

Any software using 32 bit `time_t` values, including 32 bit Linux and macOS applications and libraries, may fail when using dates later than January 18, 2038, or dates earlier than December 13, 1901.

With versions 8.0 (Visual Studio 2005) and later of Microsoft Visual C++ compilers, 32 bit Windows applications use 64 bit time_t values. If your software is compiled using a different or older compiler or C runtime library, then your 32 bit software may be subject to the limitations noted above.

PLUSNative leverages common APIs that exist in standard C libraries for date parsing and manipulation. These functions use a `time_t` data type, which typically represents the number of seconds that have passed since January 1, 1970. The maximum date allowed varies on a number of factors, including the operating system targeted, the compiler used when compiling your application, etc... Refer to your compiler and/or development framework documentation for specifics on what limitations apply to your software.

Local Validation

Local validations are validations or checks your application can perform to prevent system clock tampering. PLUSNative offers a number of pieces of information and validation objects that you can use.

API Tampering

Regardless of what programming languages and/or frameworks are used to write your application, your application will ultimately end up making calls to operating system API functions to get the system's date and time. There are some tools available (such as "Time Stopper" or "RunAsDate") on the Internet, which essentially take the place of these operating system API functions in your application (via function hooking) and send an altered date to your application. Unfortunately, this type of hack is very impractical to prevent without being very invasive. However, the good news is that detecting this type of hack is very easy, as these types of tools generally result in the underlying APIs always returning the same time. The example below shows how you can detect this kind of behavior in your application.

C/C++

```
if (SK_ERROR_NONE != SK_DateTimeValidateApi(SK_FLAGS_NONE, 0, 0, NULL))
{
   /*TODO: The operating system's date and time APIs have probably been compromised. Your
   application should take appropriate action here.*/
}
```

Evaluating Date Properties

There are several properties stored in your application's license file that you can evaluate to prevent system clock tampering. These properties include:

- `EffectiveStartDate` is the date and time which the license was issued. If the system clock predates the value stored here, the application should treat the license as being invalid or expired.
- `EffectiveEndDate` is the date and time which the license expires. Any time-limited license should only be treated as a valid license when the system clock predates or equals this property's value. If the system clock shows a date and time that occurs after the date stored in this property, the application should treat the license as being expired.
- `SignatureDate` is the date and time which SOLO Server signed the license file which was loaded by the application. If the system clock predates the value stored in this property's value, the application should treat the license as being invalid or expired.
- `LastUpdated` is the date and time which a writable license was written, and can therefore only be used with writable license files. If the system clock predates the value stored in this property's value, the application should treat the license as being invalid or expired.

To evaluate these dates, you could first create a small helper function that performs the necessary date arithmetic as shown below.

C/C++

```
BOOL IsDatePast(SK_XmlDoc license, const char *xpath, BOOL orPresent)
{
   SK_ResultCode result = SK_ERROR_NONE;
   char *value = NULL;
   int remaining = 0;
   BOOL past = FALSE;

   do
   {
      result = SK_XmlNodeGetValueDateTimeString(SK_FLAGS_NONE, license, xpath, &value);
      if (SK_ERROR_NONE != result)
      {
         /*TODO: Add error-handling code.*/
         break;
      }

      result = SK_DateTimeDaysRemaining(SK_FLAGS_NONE, value, &remaining);
      if (SK_ERROR_NONE != result)
      {
         /*TODO: Add error handling code.*/
         break;
      }

      if (TRUE == orPresent && 0 == remaining)
      {
         past = TRUE;
         break;
      }

      if (remaining < 0)
      {
         past = TRUE;
      }
   } while (FALSE);
```

```
   SK_StringDispose(SK_FLAGS_NONE, &value);
   return past;
}
```

While validating your application's license file, your application could then leverage the helper function to validate the date properties of the license file as shown in the code excerpt below.

C/C++

```
/*NOTE: This code assumes the "license" variable has been initialized using the SK_PLUS_
LicenseGetXmlDocument function.*/

/*IMPORTANT: Licenses that never expire should not validate EffectiveEndDate.*/
if (TRUE == IsDatePast(license, "/SoftwareKey/PrivateData/License/EffectiveEndDate", FALSE))
{
   /*The time-limited license has expired.*/
}

if (FALSE == IsDatePast(license, "/SoftwareKey/PrivateData/License/EffectiveStartDate",
TRUE))
{
   /*The license is not effective yet. Since this date usually represents the date in which
   the license was added in SOLO Server, this could indicate that the system clock has been
   back-dated.*/
}

if (FALSE == IsDatePast(license, "/SoftwareKey/PrivateData/License/SignatureDate", TRUE))
{
   /*The system clock's date is earlier than the date SOLO Server signed the license file.
   This suggests the system clock has been back-dated.*/
}

/*IMPORTANT: Only validate LastUpdated when using writable license files, as this date is not
present in read-only license files.*/
if (FALSE == IsDatePast(license, "/SoftwareKey/PrivateData/License/LastUpdated", TRUE))
{
   /*The system clock's date is earlier than the LastUpdated date last saved in your applic-
   ation's writable license file. This suggests the system clock has been back-dated since
   the last time your application wrote to the license file.*/
}
```

Internet Validation

When possible, using an Internet source to validate a system's clock is a good way to determine whether or not the system clock is reporting a reasonably accurate time.

SOLO Server Web Services

When your application submits requests to SOLO Server's web services (including XmlActivationService, XmlLicenseFileService, and XmlNetworkFloatingService), SOLO Server checks the requesting system's date and time. Instant SOLO Server is configured to require the requesting system's date and time to be within 24 hours of the server's date and time. If you have configured SOLO Server or use an Instant SOLO Server dedicated URL, you have

the option to **contact us** to obtain additional information on adjusting this requirement. Whenever the requesting system is outside of SOLO Server's requirement, the web service response will reflect an error with a **result code** of 5022.

C/C++

```
result = SK_CallXmlWebService(context, SK_FLAGS_NONE, SK_CONST_WEBSERVICE_
ACTIVATEINSTALLATION_URL, request, &response, &resultCode, &statusCode);
if (SK_ERROR_WEBSERVICE_RETURNED_FAILURE == result && 5022 == resultCode)
{
   //Add logic to handle the SOLO Server time validation error here.
}
```

The small excerpt of code above shows how to detect when SOLO Server time validation errors occur when calling applicable web services from PLUSNative.

Automatic Recurring Payments

SOLO Server's Payment Plans allow you to offer subscription licenses, maintenance and support subscriptions, payment over multiple installments, and more. To learn more about how to configure SOLO Server for Payment Plans, please refer to the **SOLO Server manual**.

PLUSNative: Identifying Virtual Machine Guest Environments

A good example of a Virtual Machine (VM) guest environment is where the Windows operating system is running in an isolated environment in OS X (using software like VMWare Fusion or Parallels). Although **copy protection** is designed to prevent copying a license from one machine to another, its effectiveness is often limited when the license machine happens to be a virtual machine. This is due to the fact that the virtual hardware (and the device drivers for that hardware) does not change after copying the virtual machine from one host computer to another.

License Requirements and Considerations

The way your application should react to a virtual machine guest environment will largely depend on the nature of your application and the systems on which it is run. For example, if your application is a typical desktop GUI application that should be activated once on each licensed computer, then you might opt to prevent your application from running in this type of environment. However, if your application needs to run in large office environments where employees use virtualized desktop environments, or if your application provides some kind of service (especially one which requires high-availability, such as a web application or web service), you may need to allow the application to run in these types of environments. If your application falls into the latter category, then you should take further consideration into the types of system identifiers used by your application.

Detecting Virtual Machine Guest Environments

PLUSNative makes it very easy to detect virtual machine guest environments via the `SK_SystemVirtualMachineDetect` **function**. The code below provides an example of how this can be used in your protected application(s).

C/C++

```
SK_VirtualMachineTypes vm = SK_VIRTUAL_MACHINE_NONE;

if (SK_ERROR_NONE == SK_SystemVirtualMachineDetect(context, SK_FLAGS_NONE, &vm))
{
   /*SK_SystemVirtualMachineDetect did not encounter any errors.*/
   if (SK_VIRTUAL_MACHINE_NONE == vm)
   {
      /*No virtual machine detected.*/
   }
   else
   {
     /*A virtual machine guest environment has been detected.*/
   }
}
```

Note that the example above assumes the **API context** has already been declared and initialized.

System Identification on Virtual Machine Guests

As mentioned in the introduction of this topic, **copy protection** is limited in how well it can prevent a protected application from running in a copied virtual machine guest environment. If you need license virtual machine guest environments regardless of the limitations,there are some reasonable measures you can take to help prevent this sort of thing from happening, which is what is explained here.

Algorithms

PLUSNative includes a number of system identifier algorithms that can help you better identify both physical machines and virtual machine guests. This section will explain what algorithms are most effective for virtual machine guest environments, and the behavior you can expect from each of the algorithms.

Hard Disk Volume Serial

The identifiers generated with `SK_SystemIdentifierAlgorithm.SK_SYSTEM_IDENTIFIER_ALGORITHM_HARD_DISK_VOLUME_SERIAL` use data that is generated when a partition is formatted. Since this is usually done when installing the operating system, this value is usually different between different virtual machine guests. However, as with physical machines, use of system imaging software (such as Backup Exec System Recovery) can duplicate an operating system installation, including the volume's serial number. With this limitation in mind, this algorithm is best used when accompanied by another algorithm.

Network Interface Card

The MAC (or physical) address of Network Interface Cards (NICs), which may be identified with `SK_SystemIdentifierAlgorithm.SK_SYSTEM_IDENTIFIER_ALGORITHM_NIC`, is also useful for identifying both physical machines and virtual machine guests. This is because two machines cannot have connectivity on the same network when they share the same MAC address. While this may be useful, it should be noted that most hypervisors allow you to configure the virtual machine guests to use Network Address Translation (NAT), which essentially hides the virtual machine guest behind a separate network. In other words, two virtual machines sharing the same MAC address with bridged connections will not work on the same network, but they may work on separate hosts when using NAT.

PLUSNative: Adding Support for Proxy Servers

In many environments, it may be necessary for your application to support proxy servers. This topic shows you how to set proxy server information in PLUSNative, and also summarizes some methods of automatically obtaining some of this information.

Important Note

Proxy authentication is only supported by PLUSNative with plain-HTTP requests. If you attempt to make an SSL request, PLUSNative will automatically attempt to fall-back to a plain-HTTP request (unless the **SK_FLAGS_REQUIRE_SSL flag** was specified in the **API Context** or function call). Although SSL is preferred, the web service requests are typically encrypted. If you are not encrypting the web service requests, however, this means all of the data will be transmitted as plain-text when proxy authentication is used.

Using a specified proxy server

During application initialization, you always configure an **API Context**. This the most opportune time for your application to set proxy server information using the **SK_ApiContextSetFieldString function**, as doing this will allow PLUSNative to use this information for any web service calls made with that API Context. The code you would add to configure proxy settings could resemble the following:

C/C++

```
//Your call to SK_ApiContextInitialize comes before the following code.
 //refer to the Configuring the API Context topic for instructions


SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, (SK_ApiContext_StringFields)SK_
APICONTEXT_WEBSERVICE_PROXY, "[host]:[port]");
SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, (SK_ApiContext_StringFields)SK_
APICONTEXT_WEBSERVICE_PROXYUSERNAME, "[username]");
SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, (SK_ApiContext_StringFields)SK_
APICONTEXT_WEBSERVICE_PROXYPASSWORD, "[password]");
```

Visual Basic

```
//Your call to SK_ApiContextInitialize comes before the following code.
 //refer to the Configuring the API Context topic for instructions
SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, SK_ApiContext_StringFields.SK_APICONTEXT_
WEBSERVICE_PROXY, "[host]:[port]")
SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, SK_ApiContext_StringFields.SK_APICONTEXT_
WEBSERVICE_PROXYUSERNAME, "[username]")
SK_ApiContextSetFieldString(context, SK_FLAGS_NONE, SK_ApiContext_StringFields.SK_APICONTEXT_
WEBSERVICE_PROXYPASSWORD, "[password]")
```

In the literals used in the example code excerpts above, `[host]` will reflect the host name/address of the proxy server, and `[port]` will reflect the proxy server port. Likewise, `[username]` and `[password]` will reflect the proxy authentication credentials entered by the application's end-user (and you may need to add dialogs to prompt for this information).

Proxy Auto-Configuration (PAC) scripts

Proxy Auto-Configuration (PAC) scripts are JavaScript files, which are typically interpreted by the user's web browser to automatically determine what proxy server host name/address and port should be used, if any. These files usually

reside at a URL, which can be used to download the contents of the PAC scripts.

Parsing PAC Scripts

PLUSNative does not offer any built-in support for parsing these files; however, there are several methods for doing this; however, this section is designed to help point you in the right direction.

Microsoft Windows Platforms

In Microsoft Windows, the URL can be identified by Internet Explorer settings (you can refer to the *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Internet Settings* key, which contains an *AutoConfigURL* value when one is configured for Internet Explorer) or Active Directory policy. However, parsing the key only provides the URL to the PAC script, so another option is to consider using the **`WinHttpGetProxyForUrl` function**, which will also parse the PAC script.

Apple OS X Platforms

Apple offers its own **CFProxySupport API**, which you may use for obtaining proxy configuration settings and parsing PAC scripts.

Platform Agnostic Approaches

If you need to support other platforms not mentioned above, or you wish to have a single, platform agnostic approach, then you may want to consider adding support for users to enter and store proxy configuration information in your application. If you wish to support PAC scripts via a platform agnostic approach, you could consider allowing users to enter the PAC script URL. You would then be able to use the **`SK_HttpRequest` function** to download the contents of the PAC script from the specified URL. From there, you may consider using a third-party library (such as **pacparser**) to obtain the proxy host name/address and port.

PLUSNative: Adding Support for Volume and Downloadable Licenses

Volume licenses are very permissive licenses which are designed to be used by anyone in a given organization without requiring each user to activate. Downloadable licenses with trigger code activation contain the same content as volume licenses, but these require **trigger code** validation to be authorized on each system. (This serves as a way to activate systems that need to be activated without any Internet connectivity, while avoiding the limitations of the very small payload size of trigger code activation.)

Implementing Support

To implement support for volume and downloadable licenses, you will need to change the way your application validates its system identifiers (especially if you want to support normal, activated licenses as well). The changes need to be applied prior to calling `SK_PLUS_SystemIdentifierCompare`. The code excerpt below shows how you can implement support for these kind of licenses into the LicensingSample project without affecting support for other types of licenses.

C/C++

```
wstring productOptionType = GetStringValue(L"/Soft-
wareKey/PrivateData/License/ProductOption/OptionType");
 if (0 == productOptionType.compare(L"P5VL") || //Check if this is a Protection PLUS 5 Volume
License.
   0 == productOptionType.compare(L"P5DL")) //Check if this is a Protection PLUS 5 Down-
   loadable License with Trigger Code Validation.
{
   //Volume and downloadable licenses validate identifiers differently...

   SK_XmlDoc volumeIdentifierXml = NULL;
   SK_XmlDoc licenseXml = NULL;
   char *licenseID = NULL;

   //Now grab the License ID from the license file as a string value.
   CheckResult(SK_PLUS_LicenseGetXmlDocument(m_Context, SK_FLAGS_NONE, &licenseXml));
   CheckResult(SK_XmlNodeGetValueString(SK_FLAGS_NONE, licenseXml, "/Soft-
   wareKey/PrivateData/License/LicenseID", false, &licenseID));
   SK_XmlDocumentDispose(SK_FLAGS_NONE, &licenseXml);

   //Clear the current identifiers.
   CheckResult(SK_XmlDocumentCreateFromString(SK_FLAGS_NONE, "<Activ-
   ationData><Identifiers></Identifiers></ActivationData>", &volumeIdentifierXml));
   CheckResult(SK_PLUS_SystemIdentifierCurrentSetContents(m_Context, SK_FLAGS_NONE,
   volumeIdentifierXml));
   SK_XmlDocumentDispose(SK_FLAGS_NONE, &volumeIdentifierXml);

   //Add the License ID as the current identifier.
   CheckResult(SK_PLUS_SystemIdentifierAddCurrentIdentifier(m_Context, SK_FLAGS_NONE,
   "LicenseIDIdentifier1", "LicenseIDIdentifier", (const char *)licenseID));
   SK_StringDispose(SK_FLAGS_NONE, &licenseID);
}
```

The example above only includes the logic necessary for validating a volume license. Additional logic is necessary for downloadable licenses with trigger code validation, as these licenses usually involve the use of two license files: one that is essentially the same as a volume license file, and a second, writable license file that is validated like a typical, activated license file (since it is technically activated using trigger codes).

PLUSNative: Network Floating via Semaphores

Network Floating Licensing is where an application may be restricted to running on a specific network, and restricted to a certain number of concurrent seats (where a seat may be a user or running instance of your application). This topic guides you through some basics of getting started with using network floating via semaphore files on a Windows share on a local area network (LAN). This feature has some limitations, so it is very important to review the overview of **network floating** before using it.

Getting Started

As is characteristic of PLUSNative, the first step is to **initialize an API context**. Only one network semaphore may be opened at a time for each API context initialized. In other words, if you have a single application that needs to manage separate semaphores to limit concurrent use of individual modules/features separately, each one will require its own API context.

Important

Note that the network floating via semaphore files is only supported in Windows platforms, from Windows Vista/Server 2008 and later. See the overview of **network floating** for additional details.

Before you proceed, note that you will need to establish the following pieces of information to use semaphores for network floating:

- Semaphore path: This is the path where the semaphore files are hosted. This must either be a UNC-formatted path (e.g. \\Server\Share\[Path\To\Semaphores]) or path that uses a drive letter which is mapped to a Windows share using a UNC path (e.g. X:\[Path\To\Semaphores]).
- Semaphore prefx: This is the prefix used for the semaphore file names. The samples default to "sema", which results in semaphore files being named like sema000.net, sema001.net, sema002.net, etc... By specifying a different prefix for each of your protected applications, different applications can use the same share location/path to manage their semaphore files (which can be convenient for your customers, as it can simplify configuration and management).
- Seats allowed: This is the maximum number of seats allowed (or the maximum number of semaphores that can be created). This value will usually come from your license, typically from the `/SoftwareKey/PrivateData/License/LicenseCounter` value.

Cleaning-up Orphaned Semaphores

When you attempt to open a session, the default behavior is to first get a list of files in the semaphore path, and see if any files are not present in the possible list of file names that could be present. If no files are absent/free, the default behavior is then to search for a file that may have been orphaned by attempting to obtain a lock on each semaphore file present. Since this second part of the process typically results in significant performance degradation (which worsens as the number of seats allowed increases), it is best to first try to delete all orphaned semaphore files before attempting to open one. To do this, you can call the `SK_PLUS_NetworkSemaphoreCleanup` function like:

C/C++

```
if (SK_ERROR_NONE != SK_PLUS_NetworkSemaphoreCleanup(context, SK_FLAGS_NETWORK_SEMAPHORE_
NOCLEANUPTHREAD, "[semaphorePath]", "[semaphorePrefix]", 0))
{
   //TODO: Optionally handle the error condition.
}
```

You'll notice that in the example above, we pass the `SK_FLAGS_NETWORK_SEMAPHORE_NOCLEANUPTHREAD` flag, which tells the function to only attempt to clean up the files once for now. After your semaphore has been opened

successfully, you can use this same function to create a background thread that periodically attempts to delete all orphaned files to help maximize performance.

C/C++

```
if (SK_ERROR_NONE != SK_PLUS_NetworkSemaphoreCleanup(context, SK_FLAGS_NONE, "[sem-
aphorePath]", "[semaphorePrefix]", [delay]))
{
   //TODO: Optionally handle the error condition.
}
```

In the example above, `[delay]` is the number of seconds the background thread will wait between attempts to delete orphaned files. For example, this could be replaced with 600 to try to delete files every 10 minutes. While this functionality is available as a convenience, keep in mind that this will increase network traffic for each user that has this background thread running. If you have concerns about this, consider increasing the delay to suit your needs and the needs of your customers.

Opening a Semaphore

Opening a semaphore is synonymous with attempting to allocate a seat, which is only allowed to occur if the seats are not already consumed by other users and/or application instances. A rough example of the contents of a function that opens a semaphore is shown below.

C/C++

```
if (SK_ERROR_NONE != SK_PLUS_NetworkSemaphoreOpen(context, SK_FLAGS_NONE, [seatsAllowed], "
[seatsAllowedXPath]", "[semaphorePath]", "[semaphorePrefix]", NULL, NULL))
{
   //TODO: Handle the error condition, and show the user an error message.
}
```

Of course, there are a few blanks for you to fill in for the example above to work (or even compile):

- `"[semaphorePath]"` needs to be replaced with a UNC path to the server and share that will host the semaphore files. Alternatively, you can specify a drive which is mapped to a Windows share.
- `"[semaphorePrefix]"` needs to be replaced with the semaphore filename prefix. The samples default to "sema", which results in semaphore files being named like sema000.net, sema001.net, sema002.net, etc... By specifying a different prefix for each of your protected applications, different applications can use the same share location/path to manage their semaphore files (which can be convenient for your customers, as it can simplify configuration and management).
- The number of seats/semaphores allowed can be configured one of two ways:
  - Replace `[seatsAllowed]` with the maximum number of seats allowed (or the maximum number of semaphores that can be created), and replace `"[seatsAllowedXPath]"` with a NULL or empty string.
  - If you have already successfully opened a license file before calling `SK_PLUS_NetworkSemaphoreOpen`, you may replace `[seatsAllowed]` with `0` (zero), and replace `"[seatsAllowedXPath]"` with the XPath to the license's value you wish to use as the number of maximum seats/semaphores allowed. (This is usually `"/SoftwareKey/PrivateData/License/LicenseCounter"`.)

Make sure the semaphore remains open as long as the user is accessing the feature or application being licensed.

Verifying the Semaphore

Additionally, as long as your semaphore remains open, you may validate it periodically and/or any time important features are used. The example code excerpt shown below provides a rough illustration on how you can verify your application still has a valid lock on its semaphore.

C/C++

```
if (SK_ERROR_NONE != SK_PLUS_NetworkSemaphoreVerify(context, SK_FLAGS_NONE))
{
   //TODO: Handle the error condition, and show the user an error message.
}
```

Additionally, you can get statistics on the number of seats being used for displaying the status of the license. This can be done using the `SK_PLUS_NetworkSemaphoreStatistics` function, as depicted below.

C/C++

```
int seatsAllowed = 0, seatsAllocated = 0, seatsAvailable = 0;
if (SK_ERROR_NONE != SK_PLUS_NetworkSemaphoreStatistics(context, SK_FLAGS_NONE, "[sem-
aphorePath]", "[semaphorePrefix]", &seatsAllowed, &seatsAllocated, &seatsAvailable))
{
   //TODO: Handle the error condition, and show the user an error message.
}
```

Closing the Semaphore

When your user finishes using the application or feature being licensed, you should close the semaphore to make the seat available to other users. A rough example code excerpt showing how to close a semaphore is shown below.

C/C++

```
if (SK_ERROR_NONE != SK_PLUS_NetworkSemaphoreDispose(context, SK_FLAGS_NONE))
{
   //TODO: Optionally handle the error condition.
}
```

The semaphore is also closed when disposing of the context used when opening it (when calling `SK_ApiContextDispose`). Note that, even if you give the user an option to close the semaphore, you should consider automatically closing it when the relevant application or feature is being closed to help ensure seats become available to other users as appropriate.

System Identification

System identifiers are what allow you to bind a license to a particular "system." Read more about **adding basic copy protection** for an overview on how this works and is implemented. In most cases, the system being licensed is generally a single device. However, in the case of network floating licensing, the "system" that is authorized to use the license is typically a network of devices. Naturally, this difference means system identification is usually handled differently when using network floating licensing via semaphores.

Binding the license to the share

To begin, PLUSNative includes a `SK_SYSTEM_IDENTIFIER_ALGORITHM_NETWORK_NAME` system identifier algorithm, which is designed to generate an identifier using a given server name and share name in a UNC path (e.g. \\SERVER1\\SHARE1). This algorithm can also accept a mapped network drive configured to use a UNC path and to reconnect at login.

Important

While this algorithm is import to use to prevent casual copying of licenses from one file share to another, this by itself is not enough to make things completely tamper-proof. This is because of how it is technically possible to create another server with the same name and share name on a different segment or network, and how it is possible to copy sub-directories on a share. Furthermore, these concerns are exacerbated by the ubiquity of virtualization technologies. As such, we strongly recommend using `SK_SYSTEM_IDENTIFIER_ALGORITHM_NETWORK_NAME` with additional system identifier algorithms to strengthen copy protection in your applications. We also recommend you consider using **Cloud Controlled Network Floating Licensing** instead, if possible.

Binding the license to the server's volume

A second system identifier algorithm PLUSNative includes is `SK_SYSTEM_IDENTIFIER_ALGORITHM_HARD_DISK_VOLUME_SERIAL`, which is designed to get the serial number (typically determined when a drive is formatted) of a volume/drive. While this is most often used for activating individual devices, it is also possible to use this to generate an identifier for a remote drive's volume serial. Below is some example code showing how you can implement this in PLUSNative:

C++

```
const char *volume = "R:\\";
char formatedVolumeSerialNumber[12] = { 0 };
DWORD volumeSerialNumber = 0;

GetVolumeInformation(volume, NULL, 0, &volumeSerialNumber, NULL, NULL, NULL, 0)
if (0 == volumeSerialNumber)
{
   /*TODO: Add your own error handling here!*/
}
else
{
   sprintf_s(formatedVolumeSerialNumber, sizeof(formatedVolumeSerialNumber), "%u", volumeSeri-
   alNumber);
   /*TODO: Add your error handling below for when SK_PLUS_Sys-
   temIdentifierAddCurrentIdentifier does not return SK_ERROR_NONE.*/
   SK_PLUS_SystemIdentifierAddCurrentIdentifier(context, SK_FLAGS_NONE, "HardDiskVolumeSeri-
   alIdentifier1", "HardDiskVolumeSerialIdentifier", formatedVolumeSerialNumber);
}
```

Even though this algorithm further strengthens your application's copy protection, it too is not completely tamper-proof. The volume serial number is typically determined when a drive is formatted. It is possible to duplicate this when using a system imaging tool (such as Norton Ghost) to restore a drive image backup to a new drive, copying a virtual machine to another host, or just using a third-party tool to alter a volume's serial. While these approaches do require some technical expertise, it is not outside of the capabilities of the typical IT staff who would be responsible for installing and configuring your application for a network. Consequently, we again strongly advise you to consider using additional algorithms, such as binding to application-specific data, to further strengthen copy protection for your applications.

Binding to application-specific data

In many cases, it is possible to bind a license to some information that is unique to the customer to which the license was issued. For example, if your software already has to ask the user for some sort of business license number, your software could **implement a custom identifier** that leverages that information. While this does not necessarily prevent abuse from within a given organization, it is a good way to prevent abuse from spreading outside the organization.

Binding to an absolute path

As noted in the "Binding the license to the share", binding the license to a server name and share is not enough to prevent abuse. Sometimes it is necessary to take additional measures to prevent abuse of a license from within an organization, which is not always practical to achieve by binding to application-specific data (as noted in the section immediately above). To prevent abuse on an organizations server, you can do the following:

- Ask the organization to provide the semaphore path ahead of time. Be careful to request an absolute UNC path and not a path that uses a mapped drive letter here, as this will be more reliable.
- Enter the absolute path given in the step above into either the license's custom data field or a user defined field in SOLO Server.
- Only provide the users with a License ID and password for activation (or with a license file if using volume licensing) after the above 2 steps have been completed.
- Your application's license validation logic would then be configured to require use of this path as the semaphore and license file path. If the path being used does not appear to be what is expected, your application would then display an error message that makes it clear to you (and your customer service staff) that this is why your software's license validation failed.

Review the Samples

The **sample applications** include much more functionality, including:

- Asynchronous searching for available semaphores; which can be important for high-load environments, and environments where the number of seats used often nears or reaches the maximum allowed.
- Background cleanup thread support, which helps minimize the presence of any orphaned semaphore files that may have been left behind from a computer locking up, or the application terminating abnormally.
- Complete example integration code, showing you how you can tie your network floating logic to a license, and use properties that come down from SOLO Server to make it very quick and easy to adjust things like the number of seats allowed through license refreshing (see the **ELM** overview for details).

PLUSNative Cloud-Controlled Floating Network Licensing using SOLO Server

Network Floating Licensing is where an application may be restricted to running on a specific network, and restricted to a certain number of concurrent seats (where a seat may be a user or running instance of your application). This topic guides you through some basics of getting started with using Cloud-Controlled Floating Network Licensing using SOLO Server. This feature has some limitations, so it is very important to review the overview of **network floating** before using it.

If you wish to use Cloud-Controlled Floating Network Licensing using SOLO Server, you will need to **contact us** for additional details on availability and pricing.

Getting Started

As is characteristic of PLUSNative, the first step is to **initialize an API context**. Only one network session may be opened at a time for each API context initialized. In other words, if you have a single application that needs to manage separate sessions to limit concurrent use of individual modules/features separately, each one will require its own API context.

Important

Note that, due to operating system API limitations, the Cloud-Controlled Floating Network Licensing using SOLO Server can only limit use of a session certificate file to 1 process in Windows platforms.

Opening a Session

Opening a session is synonymous with attempting to allocate a seat, which is only allowed to occur if the seats are not already consumed by other users and/or application instances. A rough example of the contents of a function that opens a session is shown below.

C/C++

```
SK_ResultCode result = SK_ERROR_NONE;
SK_XmlDoc request = NULL;
SK_XmlDoc certificate = NULL;
int resultCode = 0;
int statusCode = 0;
BOOL successful = FALSE;
char *sessionId = NULL;

do
{
   SK_PLUS_NetworkSessionDispose(context, SK_FLAGS_NONE); /*Dispose of any prior session in
   memory.*/
   if (SK_ERROR_NONE != (result = SK_SOLO_NetworkSessionOpenGetRequest(context, SK_FLAGS_
   NONE, [licenseID], "[password]", &request, NULL)))
   {
      /*TODO: Failed to generate the web service request.  Show an error message to the
      user.*/
      break;
   }

   /*Call the web service...*/
   if (SK_ERROR_NONE != (result = SK_CallXmlWebService(context, SK_FLAGS_WEBSERVICE_RAW_
```

```
   RESULT, SK_CONST_WEBSERVICE_OPENSESSION_URL, request, &certificate, &resultCode,
   &statusCode)))
   {
      if (SK_ERROR_WEBSERVICE_RETURNED_FAILURE == result)
      {
         /*TODO: The web service returned an error, so show the details about the error mes-
         sage in the response. You can use SK_XmlDocumentDecryptRsa to decrypt the response,
         and then use SK_XmlNodeGetValueString to get values from /GetNet-
         workSession/PrivateData/ResultCode and /GetNetworkSession/PrivateData/ErrorMessage.
         (See the next code sample below.)*/
      }
      else
      {
         /*TODO: Handle an error showing the details about the result returned. You can use
         SK_ApiResultCodeToString*/
      }
      break;
   }

   if (SK_ERROR_NONE != (result = SK_PLUS_NetworkSessionOpen(context, SK_FLAGS_NONE, cer-
   tificate))
   {
      /*Local validation of the certificate we received from SOLO Server failed. Show the user
      an error message, and close the session.*/
      break;
   }

   /*TODO: Make sure the value in sessionId is stored/copied to a variable that can be
   accessed later for all other requests (see the next code sample below).*/
   /*TODO: Use the SK_XmlNodeGetValue* functions to retrieve other, relevant information,
   such as the PollFrequency, PollRetryCount, PollRetryFrequency, AllocatedUntilDate,
   etc...*/

   successful = TRUE;
} while (FALSE);

SK_XmlDocumentDispose(SK_FLAGS_NONE, &request);
SK_XmlDocumentDispose(SK_FLAGS_NONE, &certificate);
SK_StringDispose(SK_ERROR_NONE, &errorMsg);

if (TRUE == successful)
{
   /*TODO: The session was opened and validated successfully. Update controls as needed.*/
}
else
{
   /*TODO: The session could not be opened.  Show an error message to the user.*/
}
```

The code excerpt above has 3 primary steps needed to open the session, and exhibits the same pattern needed to perform other actions with a session.

1. Use the `SK_SOLO_NetworkSessionOpenGetRequest` function to generate a web service request to open the session. In the example above, `[licenseId]` and `"[password]"` are replaced with the License ID and password

issued to the customer in SOLO Server.
2. Use `SK_CallXmlWebService` function to the call the web service using the request generated in the previous step.
3. Verify the response and perform the activation (in this case, open) using the `SK_PLUS_NetworkSessionOpen` function.

To avoid making this topic excessively long, the additional steps will be summarized like the list above, without also providing example code here. Of course, you can always reference the **PLUSNative samples** to get complete, functioning code that you can re-use in your applications.

Upon opening a session, you receive a network session certificate, which contains data about the session and how the protected application should behave. For example, it contains the date and time the session expires, whether or not it has been checked-out, how long it should wait between poll attempts, and more. Expanding on the example above, here is an example of how you can get the Session ID from the certificate:

C/C++

```
{
   SK_XmlDoc certificateData = NULL;
   char *sessionId = NULL;

   /*Decrypt the certificate first...*/
   if (SK_ERROR_NONE != (result = SK_XmlDocumentDecryptRsa(context, SK_FLAGS_NONE, 0, cer-
   tificate, &certificateData)))
   {
      /*TODO: Failed to decrypt and/or verify the certificate document. Show the user an
      error, and break if in the do {} while (FALSE); loop shown above.*/
   }

   if (SK_ERROR_NONE == SK_XmlNodeGetValueString(SK_FLAGS_NONE, certificateData, "/GetNet-
   workSession/PrivateData/NetworkSession/SessionID", FALSE, &sessionId))
   {
      /*TODO: Failed to get the Session ID from the certificate document. Show the user an
      error, and break if in the do {} while (FALSE); loop shown above.*/
   }

   /*TODO: Display the value of sessionId, or copy it for display later.
   /*TODO: Load other data from the certificate/response as needed.

   /*When done with this copy of the session ID, dispose of it.*/
   SK_StringDispose(SK_FLAGS_NONE, &sessionId);
   /*Dispose the decrypted certificate when done with the decrypted certificate document.*/
   SK_XmlDocumentDispose(SK_FLAGS_NONE, &certificateData);
}
```

Validating a Session

Once a session is open, and immediately after any action (other than closing the session) is performed, it is necessary to validate the session. The PLUSNative API automatically does this for you any time you call any of the `SK_PLUS_NetworkSession*` functions (though this does not apply to `SK_PLUS_NetworkSessionClose`). The automatic validation does the following:

- Makes sure we have a valid certificate that can be decrypted and passes digital signature verification.
- If a new, checked-out certificate needs to be validated, it will first load the new certificate file, decrypt it, and verify its digital signature.

- Verifies the system identifiers to ensure the certificate was issued for the system on which the protected application is running. (This prevents checked-out certificates from being copied to and successfully used on other systems which have not gone through the open and check-out process).
- Makes sure the system's local date and time is not earlier than the allocated date. Note that if the systems local clock is behind SOLO Server's time, this will generate a `SK_ERROR_LICENSE_NOT_EFFECTIVE_YET` error.
- Makes sure that the certificate has not expired by ensuring the system's local date and time is not later than the allocated until date. If the certificate is expired, this will generate a `SK_ERROR_LICENSE_EXPIRED` error.

Note that it is important to close the session rather than abandoning it when validation fails. This is because the session should not be allowed to be used when invalid, and if the session remains open on SOLO Server even though local validation has failed, abandoning it will cause SOLO Server to consider the seat to be valid until the allocated until date (or the date and time the session is set to expire if it has not polled successfully before then).

Polling a Session

When a session is open, it is set to expire by its "allocated until date." In other words, if no poll occurs before the date and time shown in the "allocated until date" is reached, the session is considered expired by SOLO Server. Successfully polling the session with SOLO Server automatically extends the allocated until date, which is what allows SOLO Server to recover orphaned sessions/seats (which is helpful for handling abnormal application termination).

1. Use the `SK_SOLO_NetworkSessionPollGetRequest` function to generate a web service request to poll the session.
2. Use `SK_CallXmlWebService` function to the call the web service using the request generated in the previous step.
3. Call the `SK_PLUS_NetworkSessionPoll` function, which verifies the response and loads the updated network session certificate.

If a session has been checked-out for extended/offline use, you may still poll to verify the session if Internet connectivity is available. However, the "allocated until date" is not automatically extended for a checked-out session.

Checking-out and Checking-in a Session

In some cases, you may find the need to allow users to check-out a session/seat for offline and/or extended use. An example could be a user who needs to use the protected application while traveling. To allow this, you can allow the user to check-out a session for a requested duration. When successful, SOLO Server will respond with an updated session certificate that has an allocated until date that is valid for the requested duration (or until the minimum or maximum check-out duration allowed if the requested duration is outside of those bounds). Follow the steps below to check-out a session:

1. Use the `SK_SOLO_NetworkSessionCheckoutGetRequest` function to generate a web service request to poll the session.
2. Use `SK_CallXmlWebService` function to the call the web service using the request generated in the previous step.
3. Call the `SK_PLUS_NetworkSessionCheckout` function, which verifies the response and loads the updated network session certificate that is configured to work without connectivity.

Once checked-out, your application may continue to poll when Internet connectivity is available (just keep in mind that attempting to poll while offline could cause performance degradation). The seat will then be consumed until the allocated until date is reached, or the user checks-in and closes the session. So in the example of a traveling user, you might allow the user to check-out for up to a week, even though the user is only expected to travel for 3 to 4 days. This allows the user to continue to use the application in a disconnected state should unforeseen circumstances (such as inclement weather) arise. If the user returns when expected, he or she could then check-in the session and return to normal connected use so the seat becomes available for other users when not in use. Follow the steps below to check-in a session that was previously checked-out:

1. Use the `SK_SOLO_NetworkSessionCheckinGetRequest` function to generate a web service request to check-in the session.
2. Use `SK_CallXmlWebService` function to the call the web service using the request generated in the previous step.

3. Call the `SK_PLUS_NetworkSessionCheckin` function, which verifies the response and loads the updated network session certificate that is configured to resume normal polling in a connected state.

Closing a Session

When your user finishes using the application or feature being licensed, you should close the session to make the seat available to other users. Follow the steps below to close a session:

1. Use the `SK_SOLO_NetworkSessionCloseGetRequest` function to generate a web service request to poll the session.
2. Use `SK_CallXmlWebService` function to the call the web service using the request generated in the previous step.
3. Call the `SK_PLUS_NetworkSessionClose` function, which verifies the response and loads the updated network session certificate.

When you're done using the PLUSNative for managing a network session you should also call the `SK_PLUS_NetworkSessionDispose` function. This function is automatically called for you when calling `SK_ApiContextDispose`, which needs to be called when you are done with the API context (such as when the application is being closed).

Review the Sample

The example code excerpts given above are only meant to provide guidance on how the APIs should be used. Review the **PLUSNative samples** to see complete, functioning example code that can be re-used in your applications.

PLUSNative: Deploying Protected Applications

Third-Party Licenses

Before redistributing PLUSNative with your application, it is important that you review the **third-party licenses**.

Distributing the PLUSNative Library

If you are linking with the PLUSNative static library, the library will be included in your executable and you should not distribute the library file. If you are linking with the PLUSNative shared dynamic-link library, you must distribute the library file with your application and install the library to the appropriate location.

Linux

It is possible to install the shared library in the application directory (which we strongly recommend), but this does require extra configuration. Installing the library to the application directory greatly simplifies deployment and also prevents other applications from installing a different version of the library that may not be compatible with your application. In order for your application to look for a shared library in the application directory, you must modify the executable and set it's *RPATH*. This can be done using the **patchelf** command. The following example demonstrates setting the *RPATH* for an executable named "program". This command would be issued in the terminal or executed from a bash script.

bash

```
patchelf --set-rpath ./ program
```

Alternatively, the PLUSNative shared library (*libPLUSNative.so*) may be installed to a location that is included in the system's default shared library search path, such as */usr/lib* or */usr/local/lib*. On 64-bit Linux distributions, the 32-bit and 64-bit libraries are generally installed to separate locations, which can vary on the distribution. This typically varies in one of two ways:

1. The 64-bit libraries are installed to *lib64*, and the 32-bit libraries are installed to *lib*.
2. The 64-bit libraries are installed to *lib64* (which is a symbolic link to *lib*), and 32-bit libraries are installed to *lib32*.

It is generally recommended that you distribute packages for the most popular Linux distributions. By targeting these, you can create packages which account for these differences.

OS X

The PLUSNative shared library (*libPLUSNative.dylib*) can be installed to the application directory (which we strongly recommend). Installing the library to the application directory greatly simplifies deployment and also prevents other applications from installing a different version of the library that may not be compatible with your application.

Since OS X does not have separate locations for 32-bit and 64-bit library files, if you choose to install the library to */usr/lib*, for example, you must create universal binary as described in the **development project references** topic) and distribute it to this location.

Windows

The PLUSNative dynamic-link library (*PLUSNative.dll*) can be installed to the application directory (which we strongly recommend). Installing the library to the application directory greatly simplifies deployment and also prevents other applications from installing a different version of the library that may not be compatible with your application.

If you opt to install to the system directory, the location varies depending on whether installing on 32-bit or 64-bit Windows. On 32-bit Windows, the library should be installed to the *system32* directory (usually

C:\Windows\system32). On 64-bit Windows, the 64-bit library should be installed to the *system32* directory (usually *C:\Windows\system32*), while the 32-bit library should be installed to the *SysWOW64* directory (usually *C:\Windows\SysWOW64*). On 64-bit Windows, Windows silently redirects 32-bit applications to the *SysWOW64* directory. In order for a 32-bit installer to correctly install 64-bit libraries to the *system32* directory the installer must be properly configured to disable redirection.

Important

When using the PLUSNative dynamic-link library (*PLUSNative.dll*) you should not distribute the PLUSNative import library (*PLUSNative_dllimport.lib*) with your application.

Installation Requirements

Protection PLUS 5 SDK license files are generally not distributed with an application. When using read-only license files, they must be issued by SOLO Server during activation. When using writable license files, such as for unmanaged evaluations/trials that do not require activation, the license files are generated the first time the application runs. Depending on where you choose to store your application's license files, you may need to set permissions on these locations during installation so that your application has write access and can save it's license files when it runs or is activated. Permissions can be set on the directory or file level. If you desire to set permissions on the file level, you can install a blank (empty) license file as a placeholder during installation.

The `SK_PermissionsGrantControlToWorld` function is provided to assist in setting such permissions, and can be used in a separate helper application you create and call from within your application's installer.

Important

Microsoft Office applications (such as Word, Excel, Access, etc...) may run in a **ClickToRun** environment. This environment has known limitations that make it problematic for licensed Office add-ins and macros to use global locations. Consequently, licensed add-ins and macros that target these environments should only use user-specific locations for licenses and aliases. See **this knowledge-base article** for more details.

Writable Licenses

When using writable licenses, PLUSNative automatically attempts to set permissions on the license file and the aliases when they are saved. You can use this to your advantage by creating a separate helper application that creates and saves the initial the license file and aliases, or add a custom command-line switch to your application that allows it to do this silently. This would enable you to call the helper application or your main application (with the command-line switch) from your installer.

SSL Certificates

It is recommended to use SSL when communicating with SOLO Server using web service calls. In order for SSL to function, the client certificate (and any intermediate certificates) for your root certificate authority (CA) must be installed on the system. On OS X, the certificates in the Keychain will be used. On Windows, the certificates in the Certificate Store will be used. On these platforms, you can generally depend on the OS to manage these certificates and there are no extra distribution requirements.

On Linux, there are a number of common locations where certificates can be found, and PLUSNative will search these locations consecutively looking for a certificate bundle to use. The location and order PLUSNative will search is listed below:

- /etc/ssl/certs/ca-certificates.crt
- /usr/share/ssl/certs/ca-bundle.crt
- /etc/pki/tls/certs/ca-bundle.crt

- /usr/share/kde4/apps/kssl/ca-bundle.crt
- /usr/share/apps/kssl/ca-bundle.crt
- /usr/share/curl/ca-bundle.crt

The first certificate bundle found will be used. If the required client certificate is not found in the certificate bundle communication over SSL will fail, even if the certificate is present in one of the other certificate bundles located on the system.

Important

PLUSNative will automatically attempt to fall-back to a plain-HTTP request (unless the **SK_FLAGS_REQUIRE_SSL** flag was specified in the **API Context** or function call). Although SSL is preferred, the web service requests are typically encrypted. If you are not encrypting the web service requests, however, this means all of the data will be transmitted as plain-text if communication over SSL fails.

Best Practices for Security and Performance

Naturally, this documentation and the samples that compliment it, is a good effort in showing you how to use Protection PLUS 5 SDK. However, there are other practices with which you need to concern yourself with in order to ensure your protected application operates both securely and efficiently.

Ask Questions

If there is something you are not entirely certain about regarding your application's security or performance, ask questions! We have plenty of resources available online that supplement this manual and can provide you further guidance, and you can always **get help** from us if you have trouble finding information.

Security

An entire manual could be dedicated to security practices; however, the focus of this section is to give you some high-level guidance meant to help you avoid some of the most common things that can lead to your application's licensing mechanisms being bypassed or compromised.

Obfuscate .NET Applications

Important

The Protection PLUS 5 SDK .NET libraries (such as PLUSManaged and PLUSManagedGui) are obfuscated, but this alone cannot prevent a hacker from modifying your application's source code. When your .NET application is compiled, it is compiled to what is called Microsoft Intermediate Language (MSIL - pronounced like "missile") code. There are tools that can be used to translate and reverse engineer MSIL code, so it is imperative that you select and use an obfuscation tool to further protect your application and your investments in it. Microsoft offers a **good summary** of this subject.

Assembly Embedding

It is possible to obfuscate your application in a way which embeds the Protection PLUS 5 SDK libraries with your application. This is recommended where possible, as it works with PLUSManaged and PLUSManagedGui, provides some extra security with most obfuscation tools, and can help simplify your deployment (as won't be necessary to distribute the library once it is embedded in your application).

Strong Name .NET Applications

Using strong naming with your .NET assemblies or applications is an easy way to help prevent hackers from altering your application, and is often already required if your application is designed to integrate with other applications (such as SharePoint). Microsoft offers a **good summary** of the benefits using strong-named assemblies.

Cryptography

There are entire books and texts on the subject of cryptography, and even various facets thereof. However, this manual offers a good summary of how Protection PLUS 5 SDK uses **cryptography**. Make sure you understand this content and how it pertains to your **licensing requirements**.

Protect your Keys

Your application will use encryption key data specified in your **SOLO Server account**. Should a hacker fully compromise this key data, it could become possible for them to arbitrarily write or rewrite your license files, which could make it possible for hackers to completely bypass your licensing logic. Consequently, it is very important for you to protect this key data in your application:

- Manually obfuscate the encryption key data in your application by storing different pieces of the key data in different areas of your code. This is a good practice, even when you already use obfuscation tools, as these tools typically only encrypt this kind of data. By breaking it out into different areas of code, you make it more difficult for the hacker to re-assemble the pieces, even if the data happens to be decrypted.
- Since your application knows all of the encryption key data needed to digitally sign writable licenses, only use a writable license if it is necessary. Since read-only licenses can only be digitally signed by SOLO Server, they are much more difficult to compromise because your application only knows enough key data to verify the signatures (but does not know enough to generate them).

Be Mindful of Input

Any point where your application is open to interaction from users is also open to interaction from malicious applications and scripts. Similarly, any data you receive from the system is data that could also be manipulated by a malicious script or program. Some simple things you can do to prevent these sorts of things from tricking your licensing include:

Validate Often (but not too often)

If your application has any critical features or functionality that are part of your core intellectual property, validate the license file in that area of code as appropriate. Make sure you only do this within reason, as you would not want this to have a negative impact on your application's performance. Furthermore, try to keep these extra checks limited to validation of your local license file, and avoid using background checking with SOLO Server over the Internet, which could result in your application appearing to be sluggish when Internet connectivity is not very fast or reliable.

Be Wary of Input

Somewhat related to the point above, it is common to have visual elements (such as buttons or menus) that run important features or functionality, and it is also common to disable those elements when the application is not licensed to use this feature. Although this does prevent a user from clicking on the element with a mouse, it is possible for another application to be used to send a "window message" to that element to emulate a successful click even when the element is disabled! While disabling elements is a good visual indicator for your users, it is important for your application checks its license to ensure it really is allowed to use the given feature when your application handles a "click" just in case.

Operating System API Tampering

Data from the system can be altered in a number of ways, ranging from altering the system's data or configuration directly, to using function hooking to trick your application into calling a hacker's code instead of the operating system's code. Without going into too much depth on how these attacks work, a good way to explain this is to provide the example of evaluating a system's time. If the system's clock has not advanced since your license was last saved, since your application started, or since just about anything was done at any point in your application, then it is fairly safe to assume the system's clock is either actively altered, or the API's being called to acquire this information from the operating system have been hooked to trick your application. Although this kind of tampering is next to impossible to prevent, the good news is that it is very easy to detect and respond to it in your protected applications, and the Protection PLUS 5 SDK APIs offer ways to easily detect this sort of activity. Furthermore, if you limit the amount of time the application or any of its features may be accessed, make sure you test for this kind of tampering throughout various parts of the application, as this will make it more difficult for hackers to develop applications that time when it alters things to try to bypass this check in your application.

Performance

Performance is another one of those subjects for which there are many books available. This section is meant to provide you some guidance on avoiding common performance issues.

File I/O

Reading and writing files can generally seem fast when you only do it once; however, if this is something that is done frequently, it can quickly add up and make your application feel sluggish, unresponsive, and clunky. Be careful about how many times your application reads and writes license files, especially since these files are heavily encrypted. Additionally, you should take particular care to consider caching license file data in memory for high-concurrency applications, such as Windows services or daemons, web applications, web services, etc...

Avoid Overkill with Security

This is somewhat related to avoiding too much file I/O. If your application reads and writes to the license file too frequently, it can drastically reduce performance. Furthermore, it is very important to make sure your application does not background check or refresh its license with SOLO Server too frequently, as this can also have a dramatic affect on your applications overall performance and could even affect the server's performance. While we would never want to understate the importance of security, it is just as important that users have a pleasant experience with your application.

Testing Protected Applications

Any time you ship your application or make it available to customers and prospects, it is pivotal to test everything thoroughly. The licensing logic added to your application should be included with this testing to help prevent unforeseen issues from arising. This topic will help guide you through this process, and may even serve as a helpful 'check-list' for you to print-out to keep track of what test has been done for new and updated applications.

Exercise Caution

Oftentimes, testing on computers that are used regularly and/or used for development can be less than ideal, can even result in unintended side effects. In general, we strongly advise you to avoid such problems by using virtualization software as described in the next section. A good example of where things can go awry when testing on development computers is the negative impacts that are possible from altering your system clock (which you might try to do when testing time-limited licenses). Some examples of how this could affect your development systems includes (but is not limited to):

- The licenses for software you have installed can begin to fail validation. This is because the licensed applications think so much time has passed that they must re-validate with the vendor's licensing servers again; however, validation with such servers is likely to fail because your system clock has been altered significantly. Visual Studio 2013 is one example which can be affected in this manner by altering your system clock, and this has the potential to be disruptive.
- Anti-virus and security tools may begin showing errors about being out-of-date.
- Automatic appointment/task reminders (in applications such as Outlook) could be shown prematurely. If, for example, these are dismissed during testing, the reminders will not be shown again when originally intended because they were already dismissed.
- When committing/checking-in source code changes into a repository, the dates on the commits could potentially be wrong. Similar issues could occur with any software that utilizes the system date.

If you cannot avoid these issues by using virtualization software, then the next best thing to do is to have a dedicated test machine which is backed-up with a complete image/snap-shot regularly.

General Testing Practices and Tools

As with any task, it is important to ensure you have the right set of tools to help ensure the task is completed well and efficiently. Here are some types of tools that we can recommend for your consideration.

Virtualization Software

Although this is not required, using virtualization software to run virtual machines is very helpful for a wide variety of reasons.

- It is possible for you to establish different virtual machines for testing different operating systems and versions thereof. For example, you could have separate virtual machines configured for testing Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, etc... Even amongst these, you could have separate virtual machines for the 32 bit and 64 bit builds of each applicable version of Windows.
- Most virtualization programs support saving "snapshots" of each virtual machine guest, which allow you to restore the virtual machine guest to the exact state it was in when the snapshot was taken. For example, if you tested installing an application on a system for the first time ever, you can revert to the snapshot when done if you need to run the test again. So if you had an issue with performing a fresh installation of your application that could not be reproduced when re-installing it, you could reproduce the problem by simply reverting back to the snapshot and starting over.
- The virtual machine guests and snapshots can be configured in a variety of different ways to cover different testing scenarios. For example, if you sell an application that also has related developer components that must be activated separately, you could test installation and usage with and without the relevant developer tools and dependencies to ensure the smoothest experience possible for all of your users.

Keep in mind that if your licensing requirements call for the detection of virtual machine guest environments to alter the way your licensing logic behaves (or even to prevent the application from running in these environments entirely), you may need to make additional considerations and adjustments for your testing procedures. These adjustments can include creating special test builds which are never publicly released, and could include testing on physical computers in place of or in addition to testing in virtual machine guests.

Helpful Information

Microsoft provides for free many Windows versions installed for many different virtual machine players/hosts. These virtual machines are setup with a 90 day evaluation of Windows.

Build and Test Automation Scripts

There are many free and commercial tools available for automating your build and testing processes. Some free tools include, but are not limited to, Power Shell scripts or batch files, shell scripts, home-grown scripts and applications you might already have in place. Having scripts related to building and testing helps avoid missing crucial steps to building applications properly, and also helps automate any mundane building and testing tasks. Additionally, **unit testing** can also help identify potential issues during development, and is a particularly convenient way of testing all possible execution paths in your code. Most integrated development environments (IDEs) have built-in features for assisting with unit testing, and many also have various code analysis features that can also help identify areas of code that could be problematic or be adjusted to follow best practices.

Establish a Testing Checklist

Since every application is different, each will have a different set of tests that should be run before releasing a new build to customers and prospects. Below is a list designed to help you get started on a checklist for testing the licensing features (and potentially other features) in your applications.

Verify your Installer

Installing

- Is the application, all of its dependencies, and any required files and permissions installed correctly? See the **PLUSManaged deployment** or **PLUSNative deployment** section for details on deploying protected applications.
- If your application should be activated once for each computer/device, does the installer prompt to obtain elevated privileges? This is required to install licenses shared with all users on a single computer, and is required to avoid problems with User Account Control (UAC) features on Windows Vista and later. This is also described in the **PLUSManaged deployment** or **PLUSNative deployment** sections.
- If your application should be activated once for every user on a computer/device, does the installer use user-centric locations? If Windows displays a prompt asking to confirm if the application installed correctly, this might indicate that the installer attempted to use global locations instead (which could indicate potential for UAC related issues).
- After installation, does the application start in the expected state? For example, if it should be a 30 day trial, does it start as such?

Upgrades

- Does the application, its dependencies, and required files get updated as appropriate?
- If the upgrade or update is free for customers who have already purchased, downloaded, installed, and activated, confirm that reactivation is not required after upgrading.
- If the upgrade is not free for customers who have already purchased, downloaded, installed, and activated, confirm that a new activation is required by your application after upgrading.
- After installing the upgrade, does the application start in the expected state? This can vary based on whether the upgrade is free or not, as described above.

Uninstalling

- Make sure the uninstaller removes the application, its dependencies, and files, as appropriate.
- If you are using a writable license file with aliases, make sure the aliases remain on the system after uninstalling. This is important so that uninstalling and reinstalling a trial does not afford users a fresh trial period, for example.

Activation and Electronic License Management

When testing activation and **Electronic License Management** (ELM) features in your application, it is important to test each possible means of performing the tasks that users are allowed/able to perform.

Using SOLO Server

You should perform the tests below with a test system that has a direct Internet connection if you allow activating online. If you allow activation through another system's Internet connection, make sure you test with manual request files while the test system lacks Internet connectivity. Furthermore, if your application supports several different types of licenses (such as full/non-expiring, time-limited, network floating, etc...), make sure you also run the tests below for each type of license.

- Try activating with a bad License ID and Password and make sure it fails, and make sure you receive a result code of 5008 from SOLO Server.
- Try activating with a License ID for a different product, and make sure you receive a result code of 5010 from SOLO Server.
- Activate with a valid License ID and Password for the correct product. Make sure it goes through successfully.
- Refresh the license.
- Deactivate the license from your application (if allowed).
- Reactivate the license using the same License ID and password. If SOLO Server is configured to allow reactivation on the same computer, you should receive the same Installation ID. Otherwise, you should receive a new Installation ID, or a result code of 5013 if no activations are remaining.
- Use the SOLO Server management interface to revoke/ban the Installation ID, and refresh the license to ensure your application revokes its license properly.
- Reactivate the license using the same License ID and password, use the SOLO Server management interface to revoke/ban the Installation ID, and deactivate the license from your application (if allowed) again. SOLO Server should return result code 5016, and your application should typically revoke its license in this circumstance as well.

Using Trigger Codes

Activation is also possible using a numeric challenge-response mechanism called *trigger codes*. Trigger codes afford your customers the convenience of being able to activate systems which lack Internet connectivity (especially systems in remote locations) by manually exchanging numeric codes with you or your customer service representatives via phone or fax.

Important

Trigger codes should only be used when absolutely necessary!

Trigger codes require the use of writable license files, which can be less secure than read-only **license files** when not using time-limited licenses.

Like any other challenge/response scheme, allowing activation via trigger codes introduces the potential to be exploited by hackers to make key generators (which are programs that issue unauthorized activations for your application) available. If the customer is able to activate using another computer or device's Internet connection, or by email, then using the manual request file is strongly recommended, as this is a much more robust and secure approach for activating disconnected devices or systems.

By definition, a trigger code is essentially just an encrypted value which gets decoded to a numeric value between 1 and 50 (which is the trigger code number). When processing the activation codes, the resulting trigger code number is used to then identify which actions should be taken in your program to alter the state of the application's license. The Protection PLUS 5 SDK **sample applications** include some basic trigger code implementations for the most common types of licenses. Make sure you run the following tests:

- Test activation with each trigger code number supported by your application. Make sure the application activates without issue, and make sure the license is altered as intended.
- If you allow activation via trigger codes while also allowing for activation with SOLO Server, you will need to test how things behave when mixing both together. For example, if you allowed users to activate time-limited licenses and non-expiring licenses:
  - Test activating a time-limited license with SOLO Server. Then activate it as a non-expiring license using a trigger code, and then refresh the license with SOLO Server. Make sure that refreshing the license does not revert it to its time-limited state. If it does, you may want to consider removing the Installation ID from the license file when the type of license (i.e. changing from time-limited to non-expiring in this example) changes due to a trigger code.
  - If trigger codes are used to activate individual application features and/or modules in a disconnected state, it will be important to ensure a refresh does not overwrite this data with old or empty data. The application should be designed so that the trigger codes update user defined fields in the license file. Prior to issuing such a trigger code to a customer or prospect, the SOLO Server license should be updated so that its user-defined fields will match what is expected to be in the license file as a result of issuing trigger codes.

Copy Protection

Your application can **identify a system** which has been activated to ensure its license is only used on the system on which it was activated. Each application can use a different combination of algorithms and requirements for this purpose, so here are some generalized tests for you to consider for each identifier algorithm being used:

- Install the application on two systems. Activate on one system, copy the activated license file to a second system. Make sure the application does not run. For applications designed for home or non-business use, this is generally a sufficient test. However, if your application is designed to be installed in large corporate environments, academic institutions, etc..., you may want to consider testing on separate computers which share very similar hardware.
- Install and activate the application on a system, and test performing typical hardware changes/upgrades to see the impact for each identifier algorithm being used. So for example, if you are using the NicIdentifierAlgorithm, you may want to test adding and removing a USB wireless network card in a laptop, if that is a reasonable scenario to expect for your application.

Time-Limited Licenses

If your application uses time-limited licenses (which are licenses that are only valid for a limited amount of time), then you should test your application's behavior thoroughly.

Important

The safest approach for testing time-limited licenses is to utilize virtualzation software. If virtualization software is not an option, then using another computer that is dedicated to testing is recommended. Examples of the potential issues you will avoid can be found in the "Exercise Caution" section at the beginning of this topic.

If your application supports multiple types of time-limited licenses (such as an evaluation and an activated, lease or subscription style license), run the tests for each type of license. Before you begin your testing, make sure your test system is not configured to synchronize with network/Internet time. In Windows, you can change this setting in the *Internet Time* tab located in the same dialog where you can adjust your system's date/time.

- Start the application, and activate it if and as appropriate (activation is often not required for evaluations).
- If your application attempts to check its status or refresh its license with SOLO Server, push the test system's date forward to the date when you expect it to start the attempts. It should fail with a result code of 5022, as the system clock will have a differ from SOLO Server's clock by more than 24 hours. Try unplugging/disconnecting the system's network adapter on the test system , and start the application again. If the refresh is not required, the second attempt should fail silently.
- If your application requires checking its status or refreshing its license with SOLO Server, push the test system's date forward to the date when you expect the requirement to be effective. Make sure the application does not run.
- Leave the network adapter disconnected, and back-date the system's clock to the correct date/time. Without Internet connectivity, alias validation should fail and your application should treat the license as being invalid or expired. Reconnect the network adapter, and test to ensure refreshing the license online (which should occur automatically) restores the license to a functioning state.

Internet Connectivity

If your application requires checking its status or refreshing its license with SOLO Server, you will need to make sure you test this functionality with and without Internet connectivity. Here are some tests you can run:

- If your application has an option, menu, or button that can cause a status check or license refresh to occur, start the application and test this functionality with and without Internet connectivity.
- If your application is configured to begin checking or refreshing its license after a period of time, push the test system's clock ahead to the date you expect checking to begin, and make sure SOLO Server returns 5022 (which will indicate the fact that SOLO Server's clock differs from the test system's clock by more than 24 hours). Also test with no Internet connectivity, and make sure the application continues without showing any errors regarding the failed attempt.
- If your application is configured to require checking or refreshing its license after a period of time, push the test system's clock ahead to the date you expect checking to begin. Otherwise, if your application always requires checking or refreshing its license, you should not need to later the test system's clock. Make sure SOLO Server returns 5022 (which will indicate the fact that SOLO Server's clock differs from the test system's clock by more than 24 hours). Also test with no Internet connectivity, and make sure the application shows an error and does not run.

Build Configurations

Sometimes it is necessary to use multiple build configurations of an application to produce different builds or releases for various reasons. Reasons could include things like branding changes, using separate builds to omit certain features that are not easily enforced through licensing code alone (i.e. an evaluation build is sometimes necessary to omit unprotected content), etc... All tests should ideally be performed against all possible build configurations, especially when code is conditionally compiled for different build configurations (typically done via preprocessor directives).

Supported Environments

All tests should be repeated for each platform supported. So for example, if your application is supported on Windows 7 and later, you would typically test your application on the 32 bit and 64 bit versions of Windows 7, Windows Server 2008 R2, Windows 8, and Windows 8 Server.

Additionally, if your application supports multiple languages, all tests should be repeated with the typical regional settings expected with each language supported.

Load Testing

Many standard desktop applications exhibit their load on systems during normal use and testing. However, if your application is expected to be run in an environment using terminal services or Citrix, or if your application includes a web application, service, or any multithreaded functionality, it is very important to test your applications performance under heavy use. For example, with web applications, you would want to avoid checking the license

status or refreshing the license file every time a page is hit. Likewise, if your application were to be used by hundreds or thousands of users on a single machine via terminal services or Citrix, frequent license validation (especially validation with SOLO Server) could congest system and network resources. Furthermore, if your application supports network floating licensing via semaphore files, it is also very important to test load test this in a realistic simulation before deploying the application to a site.

Regression Testing

As you apply changes, updates, and fixes through its lifecycle, it is important to perform **regression tests** to ensure that recent changes do not introduce new issues. Although it may not be practical to test everything each time a new build is created, it is best to automate as many tests as possible (often done via unit tests) to help prevent the introduction of new issues.

Upgrading from Protection PLUS 4

Upgrading from Protection PLUS 4 to Protection PLUS 5 SDK can be handled a number of different ways that vary based on your requirements, and the way this is handled is typically determined by the nature of how customers will upgrade the new version of your protected application.

Paid Upgrades

If your customers will be required to purchase an upgrade to the new version of your protected application, then your upgrade would most likely require the customer to activate the new version already. In this scenario, simply updating your application to use Protection PLUS 5 SDK is all that is typically necessary, as there is no need to avoid asking users to activate again. This is often the most convenient approach for upgrading from Protection PLUS 4 to Protection PLUS 5 SDK, as it is an opportunity to gain what amounts to a fresh start with the protected application's license.

Modules, Features, and Free Upgrades

There are some different approaches to consider if the new version of your protected application fits any criteria similar to the following:

- The new version of your application requires payment to upgrade while carrying-over features and modules that were activated previously.
- The new version of your application is free and does not require payment if a customer's previously activated license is still valid.

Create a Utility Program

If the updated version of your protected application uses a writable license file, it is possible for you to automate the upgrade from the Protection PLUS 4 license file to the Protection PLUS 5 SDK license file for your customers by creating a small utility program that runs during the installation of your application's update. Here is how this utility program would work using both Protection PLUS 4 and Protection PLUS 5 SDK APIs:

1. First, use the Protection PLUS 4 APIs to open and validate the old license file to ensure that it is authorized for the current computer. If the license is present and valid, then your utility program can proceed with the automatic upgrade to the new license file. Otherwise, it should skip the upgrade process so the new version of your application installs and initially runs the same way it would on a machine on which the previous version was not installed.
2. Next, look for the new license file to see if the customer has already installed or upgraded. This step is especially important if you are using a license that can be configured with additional features or modules for your application, where you might not want a user to accidentally overwrite the new version's license file when running the updated version's installer a second time. Of course, you would want to be careful to allow this to overwrite evaluation/trial licenses just in case customers used that before purchasing the upgrade to your new version. Furthermore, if your application does not offer extra complexities in its license such as additional features or modules, then this extra check might be moot, and you might find it acceptable to just always overwrite the new version's license in the case of an upgrade. You ultimately need to determine what is right for you, and you can always **get help** should you find the need for additional guidance.
3. Now that your utility program knows it is appropriate to proceed with the upgrade, the next step is to use the Protection PLUS 4 APIs to read any relevant licensing information from the old license file and store it in memory. Once again, be mindful of how you would prefer to have things work in the case of evaluation licenses. For example, it may be ideal to skip the rest this upgrade process if the old license is an evaluation license, which could allow users to obtain a fresh evaluation of the new version.
4. The next logical step is for your utility program to use the Protection PLUS 5 SDK APIs to create the license file for the new version of the application, and store the data retrieved in the previous step as appropriate. This requires that you use a writable license file, and you will also need to authorize the current computer's system identifiers so your protected application does not require activation.

5. Finally, if you wish to allow users to run both the old version and the new version of your protected application, or if you want to allow users to downgrade to the previous version, then no further action is required. Otherwise, you can use the Protection PLUS 4 APIs to remove or disable the old version's license from the system (which will cause it to require activate if the user downgrades to the previous version).

Require Activation

An alternative to creating a utility program is to require customers to activate the new version of your application. This has the benefit of being very simple, and allows you to upgrade from a Protection PLUS 4 license file to a read-only Protection PLUS 5 SDK license file.You can allow customers to activate using their existing License ID and Password through SOLO Server; however, this does have a few potential drawbacks to consider...

- It is possible that customers may have forgotten their password since purchasing. There is a way for customers to retrieve their passwords through SOLO Server's customer license portal, so this is an easy concern to address.
- Any existing licenses being reused in this manner will need to have at least one additional activation left. Especially if you have a large number of licenses eligible for a free upgrade, it may be necessary to **contact us** for assistance with incrementing the number of activations left on licenses eligible for the upgrade.
- Internet connectivity is usually required for activation (though it is possible to activate using a secondary computer or device with Internet connectivity). This could result in some additional support overhead if your customers do not generally have reliable connections.

It is also possible to leverage SOLO Server's shopping cart and **rules engine** to allow customer to acquire a free upgrade license only when they have a valid license for the previous version. However, there are fees associated with adding licenses like this in Instant SOLO Server, so please do not hesitate to **contact us** should you need any clarification on the cost associated with this option.

PLUSManaged API Reference

View the online API reference at **https://www.softwarekey.com/go/?id=29**.

PLUSNative API Overview

If your application is a native application (written in C/C++, for example), or is written in a language which can make calls to shared/dynamic-link libraries, then PLUSNative is the API designed for your needs (see more about **selecting a solution**).

The PLUSNative API reference is meant to provide very detailed documentation specifically for the PLUSNative API. Higher-level, instructional information is available, and better explains **how to use the PLUSNative API** in your applications.

Constants

Name	Description
SK_CONST_WEBSERVICE_ACTIVATEINSTALLATION_URL	Instant SOLO Server URL for the XmlActivationService web service's ActivateInstallationLicenseFile web method.
SK_CONST_WEBSERVICE_CHECKINSESSION_URL	Instant SOLO Server URL for the XmlNetworkFloatingService web service's CheckinSession web method.
SK_CONST_WEBSERVICE_CHECKINSTALLATION_URL	Instant SOLO Server URL for the XmlActivationService web service's CheckInstallationStatus web method.
SK_CONST_WEBSERVICE_CHECKOUTSESSION_URL	Instant SOLO Server URL for the XmlNetworkFloatingService web service's CheckoutSession web method.
SK_CONST_WEBSERVICE_CLOSESESSION_URL	Instant SOLO Server URL for the XmlNetworkFloatingService web service's OpenSession web method.
SK_CONST_WEBSERVICE_DEACTIVATEINSTALLATION_URL	Instant SOLO Server URL for the XmlActivationService web service's DeactivateInstallation web method.
SK_CONST_WEBSERVICE_GETLICENSEFILE_URL	Instant SOLO Server URL for the XmlLicenseFileService web service's GetLicenseFile web method.
SK_CONST_WEBSERVICE_MANUALREQUEST_URL	Instant SOLO Server URL for manually processing web service requests.
SK_CONST_WEBSERVICE_OPENSESSION_URL	Instant SOLO Server URL for the XmlNetworkFloatingService web service's OpenSession web method.
SK_CONST_WEBSERVICE_POLLSESSION_URL	Instant SOLO Server URL for the XmlNetworkFloatingService web service's PollSession web method.
SK_FLAGS_32BIT_PATHS	When specified, the function call will always use 32-bit paths/locations if applicable. This is sometimes needed to allow 32-bit and 64-bit applications share a license on the same system.
SK_FLAGS_64BIT_PATHS	When specified, the function call will always use 64-bit paths/locations if applicable. This is sometimes needed to allow 32-bit applications to initialize license files and aliases in 64 bit locations.
SK_FLAGS_APICONTEXTDISPOSE_	If specified when calling SK_ApiContextDispose, the libcrypto cryptography library will not be uninitialized.

CRYTPO_NOCLEANUP	
SK_FLAGS_APICONTEXTDISPOSE_CURL_NOCLEANUP	If specified when calling **SK_ApiContextDispose**, the libcurl HTTP library will not be uninitialized.
SK_FLAGS_APICONTEXTDISPOSE_SHUTDOWN	If specified when calling **SK_ApiContextDispose**, the PLUSNative API will shutdown and free all memory.
SK_FLAGS_APICONTEXTDISPOSE_XML2_NOCLEANUP	If specified when calling **SK_ApiContextDispose**, the libxml2 XML library will not be uninitialized.
SK_FLAGS_APICONTEXTINITIALIZE_MAINTHREAD	If specified when calling **SK_ApiContextInitialize**, any third-party dependencies will be initialized on the thread which calls this function. This flag must only be used with the first call to **SK_ApiContextInitialize**.
SK_FLAGS_APICONTEXTINITIALIZE_NOBUNDLEINIT	If specified when calling **SK_ApiContextInitialize**, the PLUSNative API will not attempt to bundle the SSL Certificates from the store into a temp folder. This flag must only be used with the first call to **SK_ApiContextInitialize**.
SK_FLAGS_DATETIME_TIME_AT_MIDNIGHT	If specified when an applicable Date-Time function is called, the operation will respect the system's date while effectively disregarding the time by always assuming midnight.
SK_FLAGS_DISABLE_WMI	When specified, the function call will not use WMI queries in Windows. While WMI works in most cases, there are occasionally cases where its use could conflict with protected applications.
SK_FLAGS_EXPLICIT_ONLY	When specified in a function call, any global flags in the **API Context** will be ignored.
SK_FLAGS_NETWORK_SEMAPHORE_NOCLEANUPTHREAD	If specified when calling **SK_PLUS_NetworkSemaphoreCleanup** no thread is created, and the function only makes a single attempt to delete orphaned semaphore files.
SK_FLAGS_NETWORK_SEMAPHORE_SKIPLOCKATTEMPT	If specified when calling **SK_PLUS_NetworkSemaphoreOpen**, the function will not attempt to lock existing files (which could be orphaned). When using this flag, it is necessary to also call **SK_PLUS_NetworkSemaphoreCleanup** to delete orphaned semaphore files; otherwise, **SK_PLUS_NetworkSemaphoreOpen** may prematurely report that no semaphores/seats are available.
SK_FLAGS_	If specified when calling **SK_PLUS_NetworkSessionLoad** or **SK_**

NETWORKSESSION_SKIPLOCKATTEMPT	**PLUS_NetworkSessionCheckout** the function will not attempt to lock the certificate file.
SK_FLAGS_NONE	Use when not specifying any flags for a given function call.
SK_FLAGS_PERMISSIONS_ALLOWEXECUTE	If specified when calling **SK_PermissionsGrantControlToWorld** against a file or directory/folder, items affected by the call will also gain execute/traverse folder privileges.
SK_FLAGS_PERMISSIONS_FOLDERINHERIT	If specified when calling **SK_PermissionsGrantControlToWorld** against a directory/folder, items created under the folder (after this call is made successfully) will inherit the new permissions.
SK_FLAGS_PERMISSIONS_NEVER_AUTO_SET	If specified when calling a function that tries to read or write a license or alias file or Registry key, the **SK_PermissionsGrantControlToWorld** is not called to automatically try to set permissions.
SK_FLAGS_REQUIRE_SSL	When specified, the function call will require Secure Sockets Layering (SSL) if applicable.
SK_FLAGS_STRING_CULTURE_USE_CURRENT	When specified, the function call will use the application's current locale rather than forcing the use of neutral culture settings are omitted.
SK_FLAGS_SYSTEMIDENTIFIER_HARDDISKVOLUMESERIALS_LEGACYOSX	If specified when calling **SK_PLUS_SystemIdentifierAlgorithmAddCurrentIdentifiers** with the **SK_SystemIdentifierAlgorithm.SK_SYSTEM_IDENTIFIER_ALGORITHM_HARD_DISK_VOLUME_SERIAL** algorithm, the legacy Mac OS X implementation will be used.
SK_FLAGS_USE_ENCRYPTION	When specified, the function call will use encryption if applicable.
SK_FLAGS_USE_SIGNATURE	When specified, the function call will use digital signatures.
SK_FLAGS_USE_SSL	When specified, the function call will use Secure Sockets Layering (SSL) if applicable.
SK_FLAGS_VALIDATE_EXPORTED_CA_CERTS	This is presently only applicable on Mac OS X (it is ignored on other platforms). If you link the dependency-free version of the static PLUSNative library for Mac OS X with your own builds of libcurl and OpenSSL, and you configure libcurl to use OpenSSL [--with-openssl] (not DarwinSSL [--with-darwinssl]) for encrypted transmissions, then you may need specify this global flag when calling **SK_ApiContextInitialize** in order to set the SSL certificate bundle path.
SK_FLAGS_WEBSERVICE_RAW_RESULT	When calling SOLO Server web services, this flag will cause **SK_**

CallXmlWebService to return the raw response (meaning the response is not decrypted for you automatically by this function). This flag is necessary to use when calling the XmlNetworkFloatingService web service with PLUSNative.

SK_CONST_WEBSERVICE_ACTIVATEINSTALLATION_URL Constant

Instant SOLO Server URL for the XmlActivationService web service's ActivateInstallationLicenseFile web method.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_ACTIVATEINSTALLATION_URL "secure.soft-
warekey.com/solo/webservices/XmlActivationService.asmx/ActivateInstallationLicenseFile"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_ACTIVATEINSTALLATION_URL As String = "secure.soft-
warekey.com/solo/webservices/XmlActivationService.asmx/ActivateInstallationLicenseFile"
```

SK_CONST_WEBSERVICE_CHECKINSESSION_URL Constant

Important

This constant is preliminary, and is subject to change.

Instant SOLO Server URL for the XmlNetworkFloatingService web service's CheckinSession web method.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_CHECKINSESSION_URL "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/CheckinSession"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_CHECKINSESSION_URL As String = "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/CheckinSession"
```

SK_CONST_WEBSERVICE_CHECKINSTALLATION_URL Constant

Instant SOLO Server URL for the XmlActivationService web service's CheckInstallationStatus web method.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_CHECKINSTALLATION_URL "secure.soft-
warekey.com/solo/webservices/XmlActivationService.asmx/CheckInstallationStatus"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_CHECKINSTALLATION_URL As String = "secure.soft-
warekey.com/solo/webservices/XmlActivationService.asmx/CheckInstallationStatus"
```

SK_CONST_WEBSERVICE_CHECKOUTSESSION_URL Constant

Important

This constant is preliminary, and is subject to change.

Instant SOLO Server URL for the XmlNetworkFloatingService web service's CheckoutSession web method.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_CHECKOUTSESSION_URL "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/CheckoutSession"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_CHECKOUTSESSION_URL As String = "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/CheckoutSession"
```

SK_CONST_WEBSERVICE_CLOSESESSION_URL Constant

Important

This constant is preliminary, and is subject to change.

Instant SOLO Server URL for the XmlNetworkFloatingService web service's OpenSession web method.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_CLOSESESSION_URL "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/CloseSession"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_CLOSESESSION_URL As String = "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/CloseSession"
```

SK_CONST_WEBSERVICE_DEACTIVATEINSTALLATION_URL Constant

Instant SOLO Server URL for the XmlActivationService web service's DeactivateInstallation web method.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_DEACTIVATEINSTALLATION_URL "secure.soft-
warekey.com/solo/webservices/XmlActivationService.asmx/DeactivateInstallation"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_DEACTIVATEINSTALLATION_URL As String = "secure.soft-
warekey.com/solo/webservices/XmlActivationService.asmx/DeactivateInstallation"
```

SK_CONST_WEBSERVICE_GETLICENSEFILE_URL Constant

Instant SOLO Server URL for the XmlLicenseFileService web service's GetLicenseFile web method.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_GETLICENSEFILE_URL "secure.soft-
warekey.com/solo/webservices/XmlLicenseFileService.asmx/GetLicenseFile"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_GETLICENSEFILE_URL As String = "secure.soft-
warekey.com/solo/webservices/XmlLicenseFileService.asmx/GetLicenseFile"
```

SK_CONST_WEBSERVICE_MANUALREQUEST_URL Constant

Instant SOLO Server URL for manually processing web service requests.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_MANUALREQUEST_URL "https://se-
cure.softwarekey.com/solo/customers/ManualRequest.aspx"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_MANUALREQUEST_URL As String = "https://se-
cure.softwarekey.com/solo/customers/ManualRequest.aspx"
```

SK_CONST_WEBSERVICE_OPENSESSION_URL Constant

Important

This constant is preliminary, and is subject to change.

Instant SOLO Server URL for the XmlNetworkFloatingService web service's OpenSession web method.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_OPENSESSION_URL "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/OpenSession"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_OPENSESSION_URL As String = "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/OpenSession"
```

SK_CONST_WEBSERVICE_POLLSESSION_URL Constant

Important

This constant is preliminary, and is subject to change.

Instant SOLO Server URL for the XmlNetworkFloatingService web service's PollSession web method.

Syntax

C/C++

```
#define SK_CONST_WEBSERVICE_POLLSESSION_URL "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/PollSession"
```

Visual Basic

```
Const SK_CONST_WEBSERVICE_POLLSESSION_URL As String = "secure.soft-
warekey.com/solo/webservices/XmlNetworkFloatingService.asmx/PollSession"
```

SK_FLAGS_32BIT_PATHS Constant

When specified, the function call will always use 32-bit paths/locations if applicable. This is sometimes needed to allow 32-bit and 64-bit applications share a license on the same system.

Syntax

C/C++

```
#define SK_FLAGS_32BIT_PATHS (0x00100000)
```

Visual Basic

```
Const SK_FLAGS_32BIT_PATHS As Long = 1048576
```

SK_FLAGS_64BIT_PATHS Constant

When specified, the function call will always use 64-bit paths/locations if applicable. This is sometimes needed to allow 32-bit applications to initialize license files and aliases in 64 bit locations.

Syntax

C/C++

```
#define SK_FLAGS_64BIT_PATHS (0x00800000)
```

Visual Basic

```
Const SK_FLAGS_64BIT_PATHS As Long = 8388608
```

SK_FLAGS_APICONTEXTDISPOSE_CRYTPO_NOCLEANUP Constant

If specified when calling **SK_ApiContextDispose**, the libcrypto cryptography library will not be uninitialized.

Syntax

C/C++

```
#define SK_FLAGS_APICONTEXTDISPOSE_CRYTPO_NOCLEANUP (0x00000002)
```

Visual Basic

```
Const SK_FLAGS_APICONTEXTDISPOSE_CRYTPO_NOCLEANUP As Long = 2
```

SK_FLAGS_APICONTEXTDISPOSE_CURL_NOCLEANUP Constant

If specified when calling **SK_ApiContextDispose**, the libcurl HTTP library will not be uninitialized.

Syntax

C/C++

```
#define SK_FLAGS_APICONTEXTDISPOSE_CURL_NOCLEANUP (0x00000004)
```

Visual Basic

```
Const SK_FLAGS_APICONTEXTDISPOSE_CURL_NOCLEANUP As Long = 4
```

SK_FLAGS_APICONTEXTDISPOSE_SHUTDOWN Constant

If specified when calling **SK_ApiContextDispose**, the PLUSNative API will shutdown and free all memory.

Remarks

Important Note

Use of this flag is not thread-safe. This should be used the last time **SK_ApiContextDispose** is called, from the application's main thread, after all other threads have finished processing.

Syntax

C/C++

```
#define SK_FLAGS_APICONTEXTDISPOSE_SHUTDOWN (0x00000001)
```

Visual Basic

```
Const SK_FLAGS_APICONTEXTDISPOSE_SHUTDOWN As Long = 1
```

SK_FLAGS_APICONTEXTDISPOSE_XML2_NOCLEANUP Constant

If specified when calling **SK_ApiContextDispose**, the libxml2 XML library will not be uninitialized.

Syntax

C/C++

```
#define SK_FLAGS_APICONTEXTDISPOSE_XML2_NOCLEANUP (0x00000008)
```

Visual Basic

```
Const SK_FLAGS_APICONTEXTDISPOSE_XML2_NOCLEANUP As Long = 8
```

SK_FLAGS_APICONTEXTINITIALIZE_MAINTHREAD Constant

If specified when calling **SK_ApiContextInitialize**, any third-party dependencies will be initialized on the thread which calls this function. This flag must only be used with the first call to **SK_ApiContextInitialize**.

Syntax

C/C++

```
#define SK_FLAGS_APICONTEXTINITIALIZE_MAINTHREAD (0x00000001)
```

Visual Basic

```
Const SK_FLAGS_APICONTEXTINITIALIZE_MAINTHREAD As Long = 1
```

SK_FLAGS_APICONTEXTINITIALIZE_NOBUNDLEINIT Constant

If specified when calling **SK_ApiContextInitialize**, the PLUSNative API will not attempt to bundle the SSL Certificates from the store into a temp folder. This flag must only be used with the first call to **SK_ApiContextInitialize**.

Syntax

C/C++

```
#define SK_FLAGS_APICONTEXTINITIALIZE_NOBUNDLEINIT (0x00000002)
```

Visual Basic

```
Const SK_FLAGS_APICONTEXTINITIALIZE_NOBUNDLEINIT As Long = 2
```

SK_FLAGS_DATETIME_TIME_AT_MIDNIGHT Constant

If specified when an applicable Date-Time function is called, the operation will respect the system's date while effectively disregarding the time by always assuming midnight.

Syntax

C/C++

```
#define SK_FLAGS_DATETIME_TIME_AT_MIDNIGHT (0x00000004)
```

Visual Basic

```
Const SK_FLAGS_DATETIME_TIME_AT_MIDNIGHT As Long = 4
```

SK_FLAGS_DISABLE_WMI Constant

When specified, the function call will not use WMI queries in Windows. While WMI works in most cases, there are occasionally cases where its use could conflict with protected applications.

Syntax

C/C++

```
#define SK_FLAGS_DISABLE_WMI (0x01000000)
```

Visual Basic

```
Const SK_FLAGS_DISABLE_WMI As Long = 16777216
```

SK_FLAGS_EXPLICIT_ONLY Constant

When specified in a function call, any global flags in the **API Context** will be ignored.

Syntax

C/C++

```
#define SK_FLAGS_EXPLICIT_ONLY (0x80000000)
```

Visual Basic

```
Const SK_FLAGS_EXPLICIT_ONLY As Long = -2147483648
```

SK_FLAGS_NETWORK_SEMAPHORE_NOCLEANUPTHREAD Constant

If specified when calling **SK_PLUS_NetworkSemaphoreCleanup** no thread is created, and the function only makes a single attempt to delete orphaned semaphore files.

Syntax

C/C++

```
#define SK_FLAGS_NETWORK_SEMAPHORE_NOCLEANUPTHREAD (0x00000002)
```

Visual Basic

```
Const SK_FLAGS_NETWORK_SEMAPHORE_NOCLEANUPTHREAD As Long = 2
```

SK_FLAGS_NETWORK_SEMAPHORE_SKIPLOCKATTEMPT Constant

If specified when calling **SK_PLUS_NetworkSemaphoreOpen**, the function will not attempt to lock existing files (which could be orphaned). When using this flag, it is necessary to also call **SK_PLUS_NetworkSemaphoreCleanup** to delete orphaned semaphore files; otherwise, **SK_PLUS_NetworkSemaphoreOpen** may prematurely report that no semaphores/seats are available.

Syntax

C/C++

```
#define SK_FLAGS_NETWORK_SEMAPHORE_SKIPLOCKATTEMPT (0x00000001)
```

Visual Basic

```
Const SK_FLAGS_NETWORK_SEMAPHORE_SKIPLOCKATTEMPT As Long = 1
```

SK_FLAGS_NETWORKSESSION_SKIPLOCKATTEMPT Constant

If specified when calling **SK_PLUS_NetworkSessionLoad** or **SK_PLUS_NetworkSessionCheckout** the function will not attempt to lock the certificate file.

Syntax

C/C++

```
#define SK_FLAGS_NETWORKSESSION_SKIPLOCKATTEMPT (0x00000001)
```

Visual Basic

```
Const SK_FLAGS_NETWORKSESSION_SKIPLOCKATTEMPT As Long = 1
```

SK_FLAGS_NONE Constant

Use when not specifying any flags for a given function call.

Remarks

Important Note

Omitting flags in a function call does not prevent global flags from being used. Use the **SK_FLAGS_EXPLICIT_ONLY** flag to omit or override global flags for a specific function call.

Syntax

C/C++

```
#define SK_FLAGS_NONE (0x00000000)
```

Visual Basic

```
Const SK_FLAGS_NONE As Long = 0
```

SK_FLAGS_PERMISSIONS_ALLOWEXECUTE Constant

If specified when calling **SK_PermissionsGrantControlToWorld** against a file or directory/folder, items affected by the call will also gain execute/traverse folder privileges.

Syntax

C/C++

```
#define SK_FLAGS_PERMISSIONS_ALLOWEXECUTE (0x00000002)
```

Visual Basic

```
Const SK_FLAGS_PERMISSIONS_ALLOWEXECUTE As Long = 2
```

SK_FLAGS_PERMISSIONS_FOLDERINHERIT Constant

If specified when calling **SK_PermissionsGrantControlToWorld** against a directory/folder, items created under the folder (after this call is made successfully) will inherit the new permissions.

Syntax

C/C++

```
#define SK_FLAGS_PERMISSIONS_FOLDERINHERIT (0x00000001)
```

Visual Basic

```
Const SK_FLAGS_PERMISSIONS_FOLDERINHERIT As Long = 1
```

SK_FLAGS_PERMISSIONS_NEVER_AUTO_SET Constant

If specified when calling a function that tries to read or write a license or alias file or Registry key, the **SK_PermissionsGrantControlToWorld** is not called to automatically try to set permissions.

Remarks

Functions which support this flag include:

- **SK_PLUS_LicenseFileLoad**
- **SK_PLUS_LicenseFileSave**
- **SK_PLUS_LicenseAliasAdd**

Syntax

C/C++

```
#define SK_FLAGS_PERMISSIONS_NEVER_AUTO_SET (0x00400000)
```

Visual Basic

```
Const SK_FLAGS_PERMISSIONS_NEVER_AUTO_SET As Long = 4194304
```

SK_FLAGS_REQUIRE_SSL Constant

When specified, the function call will require Secure Sockets Layering (SSL) if applicable.

Syntax

C/C++

```
#define SK_FLAGS_REQUIRE_SSL (0x000c0000)
```

Visual Basic

```
Const SK_FLAGS_REQUIRE_SSL As Long = 786432
```

SK_FLAGS_STRING_CULTURE_USE_CURRENT Constant

When specified, the function call will use the application's current locale rather than forcing the use of neutral culture settings are omitted.

Remarks

This flag is presently only supported in the following functions:

- **SK_XmlNodeGetValueDouble**
- **SK_XmlNodeGetValueInt**
- **SK_XmlNodeSetValueDouble**
- **SK_XmlNodeSetValueInt**

The functions listed above rely on certain standard C string manipulation and formatting routines. On POSIX-compatible platforms other than Windows and OS X, the application's locale must be set to the "POSIX" or "C" locale to function properly; however, this can have unexpected side-effects if your application uses vfork. To avoid conflicts while using vfork, you can specify this flag on any relevant functions, and manually set the locale for your application as appropriate. Note that this flag is not necessary to use on Windows or OS X.

Syntax

C/C++

```
#define SK_FLAGS_STRING_CULTURE_USE_CURRENT (0x00200000)
```

Visual Basic

```
Const SK_FLAGS_STRING_CULTURE_USE_CURRENT As Long = 2097152
```

SK_FLAGS_SYSTEMIDENTIFIER_HARDDISKVOLUMESERIALS_LEGACYOSX Constant

If specified when calling **SK_PLUS_SystemIdentifierAlgorithmAddCurrentIdentifiers** with the **SK_SystemIdentifierAlgorithm.SK_SYSTEM_IDENTIFIER_ALGORITHM_HARD_DISK_VOLUME_SERIAL** algorithm, the legacy Mac OS X implementation will be used.

Syntax

C/C++

```
#define SK_FLAGS_SYSTEMIDENTIFIER_HARDDISKVOLUMESERIALS_LEGACYOSX (0x00000001)
```

Visual Basic

```
Const SK_FLAGS_SYSTEMIDENTIFIER_HARDDISKVOLUMESERIALS_LEGACYOSX As Long = 1
```

SK_FLAGS_USE_ENCRYPTION Constant

When specified, the function call will use encryption if applicable.

Syntax

C/C++

```
#define SK_FLAGS_USE_ENCRYPTION (0x00010000)
```

Visual Basic

```
Const SK_FLAGS_USE_ENCRYPTION As Long = 65536
```

SK_FLAGS_USE_SIGNATURE Constant

When specified, the function call will use digital signatures.

Syntax

C/C++

```
#define SK_FLAGS_USE_SIGNATURE (0x00020000)
```

Visual Basic

```
Const SK_FLAGS_USE_SIGNATURE As Long = 131072
```

SK_FLAGS_USE_SSL Constant

When specified, the function call will use Secure Sockets Layering (SSL) if applicable.

Syntax

C/C++

```
#define SK_FLAGS_USE_SSL (0x00040000)
```

Visual Basic

```
Const SK_FLAGS_USE_SSL As Long = 262144
```

SK_FLAGS_VALIDATE_EXPORTED_CA_CERTS Constant

This is presently only applicable on Mac OS X (it is ignored on other platforms). If you link the dependency-free version of the static PLUSNative library for Mac OS X with your own builds of libcurl and OpenSSL, and you configure libcurl to use OpenSSL [--with-openssl] (not DarwinSSL [--with-darwinssl]) for encrypted transmissions, then you may need specify this global flag when calling **SK_ApiContextInitialize** in order to set the SSL certificate bundle path.

Syntax

C/C++

```
#define SK_FLAGS_VALIDATE_EXPORTED_CA_CERTS (0x02000000)
```

Visual Basic

```
Const SK_FLAGS_VALIDATE_EXPORTED_CA_CERTS As Long = 33554432
```

SK_FLAGS_WEBSERVICE_RAW_RESULT Constant

When calling SOLO Server web services, this flag will cause **SK_CallXmlWebService** to return the raw response (meaning the response is not decrypted for you automatically by this function). This flag is necessary to use when calling the XmlNetworkFloatingService web service with PLUSNative.

Syntax

C/C++

```
#define SK_FLAGS_WEBSERVICE_RAW_RESULT (0x00000100)
```

Visual Basic

```
Const SK_FLAGS_WEBSERVICE_RAW_RESULT As Long = 256
```

Enumerations

Name	Description
SK_ApiContext_IntFields	Field IDs used to retrieve and manipulate integer fields in an **API Context**.
SK_ApiContext_StringFields	Field IDs used to retrieve and manipulate string fields in an **API Context**.
SK_PathType	Enumeration for supported types of paths.
SK_ProductOptionType	Enumeration for supported SOLO Server product option types.
SK_RemoteSessionTypes	Enumeration for types of remote sessions.
SK_ResultCode	Possible result/return codes for the PLUSNative API.
SK_StringEncoding	Enumeration for supported types of string encodings.
SK_ SystemIdentifierAlgorithm	Enumeration for built-in system identifier algorithms.
SK_VirtualMachineTypes	Enumeration for virtual machine types/hypervisors.

SK_ApiContext_IntFields Enumeration

Field IDs used to retrieve and manipulate integer fields in an **API Context**.

Syntax

C/C++

```
typedef enum _SK_ApiContext_IntFields
{
   SK_APICONTEXT_PRODUCTID = 0,
   SK_APICONTEXT_PRODUCTOPTIONID = 1,
   SK_APICONTEXT_WEBSERVICE_CONNECTTIMEOUT = 2,
   SK_APICONTEXT_WEBSERVICE_READTIMEOUT = 3,
   SK_APICONTEXT_LICENSE_WRITABLE = 4,
   SK_APICONTEXT_NET_SESSION_DATE_TIME_THRESHOLD = 5,
   SK_APICONTEXT_NET_SESSION_VALIDATE_SYS_IDENTIFIERS = 6
} SK_ApiContext_IntFields;
```

Visual Basic

```
Public Enum SK_ApiContext_IntFields
   SK_APICONTEXT_PRODUCTID = 0
   SK_APICONTEXT_PRODUCTOPTIONID = 1
   SK_APICONTEXT_WEBSERVICE_CONNECTTIMEOUT = 2
   SK_APICONTEXT_WEBSERVICE_READTIMEOUT = 3
   SK_APICONTEXT_LICENSE_WRITABLE = 4
   SK_APICONTEXT_NET_SESSION_DATE_TIME_THRESHOLD = 5
   SK_APICONTEXT_NET_SESSION_VALIDATE_SYS_IDENTIFIERS = 6
End Enum
```

Values

`SK_APICONTEXT_PRODUCTID (0)`

Product ID (from SOLO Server) of the application.

`SK_APICONTEXT_PRODUCTOPTIONID (1)`

Product Option ID (from SOLO Server) of the application. Leave/set to zero (0) to disregard the Product Option ID during activation.

`SK_APICONTEXT_WEBSERVICE_CONNECTTIMEOUT (2)`

The amount of time (in seconds) to wait while trying to connect. Set to zero (0) to wait indefinitely.

`SK_APICONTEXT_WEBSERVICE_READTIMEOUT (3)`

The amount of time (in seconds) to wait while reading/downloading a web service response. Set to zero (0) to wait indefinitely.

`SK_APICONTEXT_LICENSE_WRITABLE (4)`

Set to TRUE (1) to use a writable license or FALSE (0) (which is the default) to use a read-only license.

`SK_APICONTEXT_NET_SESSION_DATE_TIME_THRESHOLD (5)`

Used with network floating sessions this specifies the allowed amount of difference in time (in minutes) between the licensed system's time and SOLO Server's time.

`SK_APICONTEXT_NET_SESSION_VALIDATE_SYS_IDENTIFIERS (6)`

Used with network floating sessions. Set to TRUE (1) to validate the network session system identifiers. Set to FALSE (0) to omit the network session system identifiers validation.

SK_ApiContext_StringFields Enumeration

Field IDs used to retrieve and manipulate string fields in an **API Context**.

Syntax

C/C++

```
typedef enum _SK_ApiContext_StringFields
{
   SK_APICONTEXT_PRODUCTVERSION = 0,
   SK_APICONTEXT_ENVELOPEKEY = 1,
   SK_APICONTEXT_WEBSERVICE_PROXY = 2,
   SK_APICONTEXT_WEBSERVICE_PROXYUSERNAME = 3,
   SK_APICONTEXT_WEBSERVICE_PROXYPASSWORD = 4
} SK_ApiContext_StringFields;
```

Visual Basic

```
Public Enum SK_ApiContext_StringFields
   SK_APICONTEXT_PRODUCTVERSION = 0
   SK_APICONTEXT_ENVELOPEKEY = 1
   SK_APICONTEXT_WEBSERVICE_PROXY = 2
   SK_APICONTEXT_WEBSERVICE_PROXYUSERNAME = 3
   SK_APICONTEXT_WEBSERVICE_PROXYPASSWORD = 4
End Enum
```

Values

`SK_APICONTEXT_PRODUCTVERSION (0)`

The version of the application/product. (Must be formatted like x.x.x.x where each 'x' represents a positive integer value no larger than 5 digits in length.)

`SK_APICONTEXT_ENVELOPEKEY (1)`

The decryption key for the encryption key envelope (from SOLO Server).

`SK_APICONTEXT_WEBSERVICE_PROXY (2)`

Optional proxy server information used when calling web services (must be formatted like [host]:[port]).

`SK_APICONTEXT_WEBSERVICE_PROXYUSERNAME (3)`

Optional username used for proxy server authentication when calling web services.

`SK_APICONTEXT_WEBSERVICE_PROXYPASSWORD (4)`

Optional password used for proxy server authentication when calling web services.

SK_PathType Enumeration

Enumeration for supported types of paths.

Syntax

C/C++

```
typedef enum _SK_PathType
{
   SK_PATH_TYPE_FILE = 0,
   SK_PATH_TYPE_REGISTRY = 1,
   SK_PATH_TYPE_FOLDER = 2,
   SK_PATH_TYPE_IMAGE = 3
} SK_PathType;
```

Visual Basic

```
Public Enum SK_PathType
   SK_PATH_TYPE_FILE = 0
   SK_PATH_TYPE_REGISTRY = 1
   SK_PATH_TYPE_FOLDER = 2
   SK_PATH_TYPE_IMAGE = 3
End Enum
```

Values

SK_PATH_TYPE_FILE (0)

A file on the local system's file system.

SK_PATH_TYPE_REGISTRY (1)

A registry key value on the local system.

SK_PATH_TYPE_FOLDER (2)

A directory/folder on the local system's file system.

SK_PATH_TYPE_IMAGE (3)

An image on the local system's file system.

SK_ProductOptionType Enumeration

Enumeration for supported SOLO Server product option types.

Syntax

C/C++

```
typedef enum _SK_ProductOptionType
{
   SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE = 0,
   SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE_WITH_QUANTITY = 1,
   SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE_WITH_FIXEDVALUE = 2,
   SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE_WITH_DAYSLEFT = 3,
   SK_PRODUCT_OPTION_TYPE_VOLUME_LICENSE = 4,
   SK_PRODUCT_OPTION_TYPE_DOWNLOADABLE_LICENSE_WITH_TRIGGERCODE_VALIDATION = 5,
   SK_PRODUCT_OPTION_TYPE_CUSTOM = 6
} SK_ProductOptionType;
```

Visual Basic

```
Public Enum SK_ProductOptionType
   SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE = 0
   SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE_WITH_QUANTITY = 1
   SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE_WITH_FIXEDVALUE = 2
   SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE_WITH_DAYSLEFT = 3
   SK_PRODUCT_OPTION_TYPE_VOLUME_LICENSE = 4
   SK_PRODUCT_OPTION_TYPE_DOWNLOADABLE_LICENSE_WITH_TRIGGERCODE_VALIDATION = 5
   SK_PRODUCT_OPTION_TYPE_CUSTOM = 6
End Enum
```

Values

`SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE (0)`

Standard Protection PLUS 4 and 5 activation code.

`SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE_WITH_QUANTITY (1)`

Protection PLUS 4 and 5 activation code with the quantity of items ordered.

`SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE_WITH_FIXEDVALUE (2)`

Protection PLUS 4 and 5 activation code with an additional fixed value.

`SK_PRODUCT_OPTION_TYPE_ACTIVATIONCODE_WITH_DAYSLEFT (3)`

Protection PLUS 4 and 5 activation code with the number of days left until the license expires.

`SK_PRODUCT_OPTION_TYPE_VOLUME_LICENSE (4)`

Protection PLUS 5 volume license.

`SK_PRODUCT_OPTION_TYPE_DOWNLOADABLE_LICENSE_WITH_TRIGGERCODE_VALIDATION (5)`

Protection PLUS 5 downloadable license with Protection PLUS 4 compatible trigger code validation.

`SK_PRODUCT_OPTION_TYPE_CUSTOM (6)`

An unknown or custom value. Read the **OptionType field** to get this value.

SK_RemoteSessionTypes Enumeration

Enumeration for types of remote sessions.

Syntax

C/C++

```
typedef enum _SK_RemoteSessionTypes
{
   SK_REMOTE_SESSION_NONE = 0,
   SK_REMOTE_SESSION_TERMINAL_SERVICES = 1
} SK_RemoteSessionTypes;
```

Visual Basic

```
Public Enum SK_RemoteSessionTypes
   SK_REMOTE_SESSION_NONE = 0
   SK_REMOTE_SESSION_TERMINAL_SERVICES = 1
End Enum
```

Values

SK_REMOTE_SESSION_NONE (0)

None

SK_REMOTE_SESSION_TERMINAL_SERVICES (1)

Terminal Services / Remote Desktop

SK_ResultCode Enumeration

Possible result/return codes for the PLUSNative API.

Remarks

The latest **Protection PLUS 5 SDK result codes** are documented online. The latest **SOLO Server result codes** (returned by SOLO Server web service method calls) are also documented online.

Syntax

C/C++

```
typedef enum _SK_ResultCode
{
   SK_ERROR_NONE = 0,
   SK_ERROR_INVALID_DATA = 9001,
   SK_ERROR_INVALID_SERVER_KEY = 9002,
   SK_ERROR_INVALID_CLIENT_KEY = 9003,
   SK_ERROR_DECRYPTION_FAILED = 9004,
   SK_ERROR_VERIFICATION_FAILED = 9005,
   SK_ERROR_ENCRYPTION_FAILED = 9006,
   SK_ERROR_SIGNING_FAILED = 9007,
   SK_ERROR_SESSION_VERIFICATION_FAILED = 9008,
   SK_ERROR_INSTALLATIONID_REQUIRED = 9009,
   SK_ERROR_TRIGGER_CODE_INVALID = 9010,
   SK_ERROR_TRIGGER_CODE_EVENT_DATA_INVALID = 9011,
   SK_ERROR_INVALID_LICENSE_TYPE = 9012,
   SK_ERROR_XML_PARSER_FAILED = 9013,
   SK_ERROR_XML_NODE_MISSING = 9014,
   SK_ERROR_INVALID_ARGUMENTS = 9015,
   SK_ERROR_CONTEXT_INVALID = 9016,
   SK_ERROR_STRING_CONVERSION_FAILED = 9017,
   SK_ERROR_DATETIME_CONVERSION_FAILED = 9018,
   SK_ERROR_PLUS_EVALUATION_WARNING = 9019,
   SK_ERROR_PLUS_EVALUATION_INVALID = 9020,
   SK_ERROR_INVALID_PRODUCTID = 9021,
   SK_ERROR_INVALID_PRODUCTOPTIONID = 9022,
   SK_ERROR_ENVELOPE_TYPE_INVALID = 9023,
   SK_ERROR_INSUFFICIENT_IMAGE_SIZE = 9024,
   SK_ERROR_INVALID_IMAGE = 9025,
   SK_ERROR_WEBSERVICE_INVALID_CONFIGURATION = 9100,
   SK_ERROR_WEBSERVICE_CALL_FAILED = 9101,
   SK_ERROR_WEBSERVICE_RETURNED_FAILURE = 9102,
   SK_ERROR_REQUIRED_SERVER_VALIDATION_FAILED = 9103,
   SK_ERROR_HTTP_INITIALIZATION_FAILED = 9104,
   SK_ERROR_HTTP_CONNECTION_FAILED = 9105,
   SK_ERROR_HTTP_COULD_NOT_RESOLVE_HOST = 9106,
   SK_ERROR_SSL_FAILED = 9107,
   SK_ERROR_COULD_NOT_LOAD_LICENSE = 9200,
   SK_ERROR_COULD_NOT_SAVE_LICENSE = 9201,
   SK_ERROR_LICENSE_NOT_EFFECTIVE_YET = 9202,
   SK_ERROR_LICENSE_EXPIRED = 9203,
   SK_ERROR_LICENSE_ALIAS_VALIDATION_FAILED = 9204,
```

```
   SK_ERROR_LICENSE_ALIAS_VALIDATION_TIME_MISMATCH = 9205,
   SK_ERROR_COULD_NOT_SAVE_NETWORK_CERTIFICATE = 9206,
   SK_ERROR_NETWORK_CERTIFICATE_INVALID_PATH = 9207,
   SK_ERROR_NETWORK_CERTIFICATE_REQUIRED = 9208,
   SK_ERROR_COULD_NOT_DELETE_FILE = 9209,
   SK_ERROR_NETWORK_SEMAPHORE_INVALID_PATH = 9210,
   SK_ERROR_NETWORK_LICENSE_FULL = 9211,
   SK_ERROR_NETWORK_SEMAPHORE_LOCK_FAILED = 9212,
   SK_ERROR_MODULE_NOT_ACTIVE = 9213,
   SK_ERROR_COULD_NOT_OPEN_FILE = 9214,
   SK_ERROR_COULD_NOT_READ_FILE = 9215,
   SK_ERROR_COULD_NOT_WRITE_FILE = 9216,
   SK_ERROR_COULD_NOT_OPEN_REGISTRY_KEY = 9217,
   SK_ERROR_COULD_NOT_READ_REGISTRY_KEY = 9218,
   SK_ERROR_COULD_NOT_WRITE_REGISTRY_KEY = 9219,
   SK_ERROR_IO_OPERATION_FAILED = 9220,
   SK_ERROR_COULD_NOT_READ_PERMISSIONS = 9221,
   SK_ERROR_COULD_NOT_SET_PERMISSIONS = 9222,
   SK_ERROR_SSL_CERTIFICATE_EXPORT_FAILED = 9223,
   SK_ERROR_SSL_CERTIFICATE_UNAVAILABLE = 9224,
   SK_ERROR_COULD_NOT_LOAD_VOLUME_DOWNLOADABLE_LICENSE = 9225,
   SK_ERROR_SYSTEM_TIME_VERIFICATION_FAILED = 9300,
   SK_ERROR_SYSTEM_TIME_INVALID = 9301,
   SK_ERROR_VIRTUAL_MACHINE_DETECTED = 9302,
   SK_ERROR_REMOTE_SESSION_DETECTED = 9303,
   SK_ERROR_LICENSE_SYSTEM_IDENTIFIERS_DONT_MATCH = 9400,
   SK_ERROR_PLATFORM_ERROR = 9401,
   SK_ERROR_UNSUPPORTED_OS = 9402,
   SK_ERROR_MEMORY_ALLOCATION = 9403,
   SK_ERROR_LIBRARY_UNAVAILABLE = 9404,
   SK_ERROR_LIBRARY_FUNCTION_UNAVAILABLE = 9405
} SK_ResultCode;
```

Visual Basic

```
Public Enum SK_ResultCode
   SK_ERROR_NONE = 0
   SK_ERROR_INVALID_DATA = 9001
   SK_ERROR_INVALID_SERVER_KEY = 9002
   SK_ERROR_INVALID_CLIENT_KEY = 9003
   SK_ERROR_DECRYPTION_FAILED = 9004
   SK_ERROR_VERIFICATION_FAILED = 9005
   SK_ERROR_ENCRYPTION_FAILED = 9006
   SK_ERROR_SIGNING_FAILED = 9007
   SK_ERROR_SESSION_VERIFICATION_FAILED = 9008
   SK_ERROR_INSTALLATIONID_REQUIRED = 9009
   SK_ERROR_TRIGGER_CODE_INVALID = 9010
   SK_ERROR_TRIGGER_CODE_EVENT_DATA_INVALID = 9011
   SK_ERROR_INVALID_LICENSE_TYPE = 9012
   SK_ERROR_XML_PARSER_FAILED = 9013
   SK_ERROR_XML_NODE_MISSING = 9014
   SK_ERROR_INVALID_ARGUMENTS = 9015
   SK_ERROR_CONTEXT_INVALID = 9016
   SK_ERROR_STRING_CONVERSION_FAILED = 9017
   SK_ERROR_DATETIME_CONVERSION_FAILED = 9018
   SK_ERROR_PLUS_EVALUATION_WARNING = 9019
   SK_ERROR_PLUS_EVALUATION_INVALID = 9020
   SK_ERROR_INVALID_PRODUCTID = 9021
   SK_ERROR_INVALID_PRODUCTOPTIONID = 9022
   SK_ERROR_ENVELOPE_TYPE_INVALID = 9023
   SK_ERROR_INSUFFICIENT_IMAGE_SIZE = 9024
   SK_ERROR_INVALID_IMAGE = 9025
   SK_ERROR_WEBSERVICE_INVALID_CONFIGURATION = 9100
   SK_ERROR_WEBSERVICE_CALL_FAILED = 9101
   SK_ERROR_WEBSERVICE_RETURNED_FAILURE = 9102
   SK_ERROR_REQUIRED_SERVER_VALIDATION_FAILED = 9103
   SK_ERROR_HTTP_INITIALIZATION_FAILED = 9104
   SK_ERROR_HTTP_CONNECTION_FAILED = 9105
   SK_ERROR_HTTP_COULD_NOT_RESOLVE_HOST = 9106
   SK_ERROR_SSL_FAILED = 9107
   SK_ERROR_COULD_NOT_LOAD_LICENSE = 9200
   SK_ERROR_COULD_NOT_SAVE_LICENSE = 9201
   SK_ERROR_LICENSE_NOT_EFFECTIVE_YET = 9202
   SK_ERROR_LICENSE_EXPIRED = 9203
   SK_ERROR_LICENSE_ALIAS_VALIDATION_FAILED = 9204
   SK_ERROR_LICENSE_ALIAS_VALIDATION_TIME_MISMATCH = 9205
   SK_ERROR_COULD_NOT_SAVE_NETWORK_CERTIFICATE = 9206
   SK_ERROR_NETWORK_CERTIFICATE_INVALID_PATH = 9207
   SK_ERROR_NETWORK_CERTIFICATE_REQUIRED = 9208
   SK_ERROR_COULD_NOT_DELETE_FILE = 9209
   SK_ERROR_NETWORK_SEMAPHORE_INVALID_PATH = 9210
   SK_ERROR_NETWORK_LICENSE_FULL = 9211
   SK_ERROR_NETWORK_SEMAPHORE_LOCK_FAILED = 9212
   SK_ERROR_MODULE_NOT_ACTIVE = 9213
   SK_ERROR_COULD_NOT_OPEN_FILE = 9214
   SK_ERROR_COULD_NOT_READ_FILE = 9215
   SK_ERROR_COULD_NOT_WRITE_FILE = 9216
```

```
    SK_ERROR_COULD_NOT_OPEN_REGISTRY_KEY = 9217
    SK_ERROR_COULD_NOT_READ_REGISTRY_KEY = 9218
    SK_ERROR_COULD_NOT_WRITE_REGISTRY_KEY = 9219
    SK_ERROR_IO_OPERATION_FAILED = 9220
    SK_ERROR_COULD_NOT_READ_PERMISSIONS = 9221
    SK_ERROR_COULD_NOT_SET_PERMISSIONS = 9222
    SK_ERROR_SSL_CERTIFICATE_EXPORT_FAILED = 9223
    SK_ERROR_SSL_CERTIFICATE_UNAVAILABLE = 9224
    SK_ERROR_COULD_NOT_LOAD_VOLUME_DOWNLOADABLE_LICENSE = 9225
    SK_ERROR_SYSTEM_TIME_VERIFICATION_FAILED = 9300
    SK_ERROR_SYSTEM_TIME_INVALID = 9301
    SK_ERROR_VIRTUAL_MACHINE_DETECTED = 9302
    SK_ERROR_REMOTE_SESSION_DETECTED = 9303
    SK_ERROR_LICENSE_SYSTEM_IDENTIFIERS_DONT_MATCH = 9400
    SK_ERROR_PLATFORM_ERROR = 9401
    SK_ERROR_UNSUPPORTED_OS = 9402
    SK_ERROR_MEMORY_ALLOCATION = 9403
    SK_ERROR_LIBRARY_UNAVAILABLE = 9404
    SK_ERROR_LIBRARY_FUNCTION_UNAVAILABLE = 9405
End Enum
```

Values

SK_ERROR_NONE (0)

No error.

SK_ERROR_INVALID_DATA (9001)

The presence of invalid data has been detected.

SK_ERROR_INVALID_SERVER_KEY (9002)

The presence of an invalid server key has been detected.

Only used in PLUSManaged.

SK_ERROR_INVALID_CLIENT_KEY (9003)

The presence of an invalid client key has been detected.

Only used in PLUSManaged.

SK_ERROR_DECRYPTION_FAILED (9004)

The requested decryption operation has failed.

SK_ERROR_VERIFICATION_FAILED (9005)

The requested verification operation has failed.

SK_ERROR_ENCRYPTION_FAILED (9006)

The requested encryption operation has failed

SK_ERROR_SIGNING_FAILED (9007)

The requested signing operation has failed.

SK_ERROR_SESSION_VERIFICATION_FAILED (9008)

The requested session code verification has failed.

SK_ERROR_INSTALLATIONID_REQUIRED (9009)

An Installation ID is required but is not present.

SK_ERROR_TRIGGER_CODE_INVALID (9010)

An invalid "Activation Code 1" value was entered by the user.

SK_ERROR_TRIGGER_CODE_EVENT_DATA_INVALID (9011)

An invalid "Activation Code 2" value was entered by the user.

SK_ERROR_INVALID_LICENSE_TYPE (9012)

The license type is invalid or not supported.

SK_ERROR_XML_PARSER_FAILED (9013)

The XML parser encountered an error.

Only used in PLUSNative.

SK_ERROR_XML_NODE_MISSING (9014)

The requested XML node could not be found.

Only used in PLUSNative.

SK_ERROR_INVALID_ARGUMENTS (9015)

Some or all of the arguments are invalid.

Only used in PLUSNative.

SK_ERROR_CONTEXT_INVALID (9016)

The API context passed into the function call is not valid.

Only used in PLUSNative.

SK_ERROR_STRING_CONVERSION_FAILED (9017)

A string conversion operation failed.

Only used in PLUSNative.

SK_ERROR_DATETIME_CONVERSION_FAILED (9018)

A date-time conversion operation failed.
Only used in PLUSNative.

SK_ERROR_PLUS_EVALUATION_WARNING (9019)

No error actually occurred but that an evaluation envelope is being used.

SK_ERROR_PLUS_EVALUATION_INVALID (9020)

The Protection PLUS 5 SDK evaluation is invalid or expired.

SK_ERROR_INVALID_PRODUCTID (9021)

The Product ID is not valid.

SK_ERROR_INVALID_PRODUCTOPTIONID (9022)

The Product Option ID is not valid.

SK_ERROR_ENVELOPE_TYPE_INVALID (9023)

The envelope is not valid.

SK_ERROR_INSUFFICIENT_IMAGE_SIZE (9024)

The license image is either too small to hold the license data or is under the minimum image size required.

SK_ERROR_INVALID_IMAGE (9025)

The license image is not valid.

SK_ERROR_WEBSERVICE_INVALID_CONFIGURATION (9100)

The configuration of the requested Web Service is invalid.
Only used in PLUSManaged.

SK_ERROR_WEBSERVICE_CALL_FAILED (9101)

An unexpected failure occurred during an attempt to call a Web Service.

SK_ERROR_WEBSERVICE_RETURNED_FAILURE (9102)

A call to a Web Service succeeded but the functionality of the Web Service returned an indicator of failure.

SK_ERROR_REQUIRED_SERVER_VALIDATION_FAILED (9103)

Validation against SOLO Server is required but could not be completed.

SK_ERROR_HTTP_INITIALIZATION_FAILED (9104)

The HTTP client failed to initialize.
Only used in PLUSNative.

SK_ERROR_HTTP_CONNECTION_FAILED (9105)

The server could not be reached. Verify that the computer is connected to the Internet and that the fire-wall/proxy is set-up properly.

Only used in PLUSNative.

`SK_ERROR_HTTP_COULD_NOT_RESOLVE_HOST (9106)`

The server could not be located. Verify that the computer is connected to the Internet and that the fire-wall/proxy is set-up properly.

Only used in PLUSNative.

`SK_ERROR_SSL_FAILED (9107)`

The HTTPS request failed due to an SSL related error.

Only used in PLUSNative.

`SK_ERROR_COULD_NOT_LOAD_LICENSE (9200)`

License could not be loaded.

`SK_ERROR_COULD_NOT_SAVE_LICENSE (9201)`

License could not be saved.

`SK_ERROR_LICENSE_NOT_EFFECTIVE_YET (9202)`

License is not yet effective.

`SK_ERROR_LICENSE_EXPIRED (9203)`

License has expired.

`SK_ERROR_LICENSE_ALIAS_VALIDATION_FAILED (9204)`

Validation of license alias has failed.

`SK_ERROR_LICENSE_ALIAS_VALIDATION_TIME_MISMATCH (9205)`

Validation of license alias time has failed due to mismatch.

`SK_ERROR_COULD_NOT_SAVE_NETWORK_CERTIFICATE (9206)`

Network certificate could not be saved.

Only used in PLUSManaged.

`SK_ERROR_NETWORK_CERTIFICATE_INVALID_PATH (9207)`

The Network Certificate path does not match the path specified during checkout.

Only used in PLUSManaged.

`SK_ERROR_NETWORK_CERTIFICATE_REQUIRED (9208)`

A valid network session certificate is required but is not present.
Only used in PLUSManaged.

SK_ERROR_COULD_NOT_DELETE_FILE (9209)

Could not delete file.

SK_ERROR_NETWORK_SEMAPHORE_INVALID_PATH (9210)

The network path is not valid or is unavailable.

SK_ERROR_NETWORK_LICENSE_FULL (9211)

The number of allowed concurrent users has been reached.

SK_ERROR_NETWORK_SEMAPHORE_LOCK_FAILED (9212)

Failed to create network semaphore.

SK_ERROR_MODULE_NOT_ACTIVE (9213)

The activation was successful; however another activation is required to enable use of this application.

SK_ERROR_COULD_NOT_OPEN_FILE (9214)

An attempt to open a file failed.
Only used in PLUSNative.

SK_ERROR_COULD_NOT_READ_FILE (9215)

An attempt to read a file failed.
Only used in PLUSNative.

SK_ERROR_COULD_NOT_WRITE_FILE (9216)

An attempt to write a file failed.
Only used in PLUSNative.

SK_ERROR_COULD_NOT_OPEN_REGISTRY_KEY (9217)

An attempt to open a registry key failed.
Only used in PLUSNative.

SK_ERROR_COULD_NOT_READ_REGISTRY_KEY (9218)

An attempt to read a registry key value failed.
Only used in PLUSNative.

SK_ERROR_COULD_NOT_WRITE_REGISTRY_KEY (9219)

An attempt to write a registry key value failed.
Only used in PLUSNative.

`SK_ERROR_IO_OPERATION_FAILED (9220)`

An attempt to perform an I/O operation failed.
Only used in PLUSNative.

`SK_ERROR_COULD_NOT_READ_PERMISSIONS (9221)`

An attempt to read a file or registry key's permissions failed.
Only used in PLUSNative.

`SK_ERROR_COULD_NOT_SET_PERMISSIONS (9222)`

An attempt to set a file or registry key's permissions failed.
Only used in PLUSNative.

`SK_ERROR_SSL_CERTIFICATE_EXPORT_FAILED (9223)`

Failed to export the SSL client certificate bundle.
Only used in PLUSNative.

`SK_ERROR_SSL_CERTIFICATE_UNAVAILABLE (9224)`

The client certificate for SSL communication could not be found.
Only used in PLUSNative.

`SK_ERROR_COULD_NOT_LOAD_VOLUME_DOWNLOADABLE_LICENSE (9225)`

The volume or downloadable license file could not be found or loaded.

`SK_ERROR_SYSTEM_TIME_VERIFICATION_FAILED (9300)`

Verification of system time has failed.

`SK_ERROR_SYSTEM_TIME_INVALID (9301)`

System time is not valid.

`SK_ERROR_VIRTUAL_MACHINE_DETECTED (9302)`

The application determined it is running in a virtual machine.

`SK_ERROR_REMOTE_SESSION_DETECTED (9303)`

The application determined it is running in a remote session.

`SK_ERROR_LICENSE_SYSTEM_IDENTIFIERS_DONT_MATCH (9400)`

License system identifiers do not match.

`SK_ERROR_PLATFORM_ERROR (9401)`

Platform specific API or system call fails.

Only used in PLUSNative.

`SK_ERROR_UNSUPPORTED_OS (9402)`

The current operating system is not supported by this feature or function.

`SK_ERROR_MEMORY_ALLOCATION (9403)`

Memory could not be allocated.

Only used in PLUSNative.

`SK_ERROR_LIBRARY_UNAVAILABLE (9404)`

Required system library is missing for failed to load.

Only used in PLUSNative.

`SK_ERROR_LIBRARY_FUNCTION_UNAVAILABLE (9405)`

Required library function is missing.

Only used in PLUSNative.

SK_StringEncoding Enumeration

Enumeration for supported types of string encodings.

Syntax

C/C++

```
typedef enum _SK_StringEncoding
{
   SK_STRING_ENCODING_ANSI = 0,
   SK_STRING_ENCODING_UTF8 = 1
} SK_StringEncoding;
```

Visual Basic

```
Public Enum SK_StringEncoding
   SK_STRING_ENCODING_ANSI = 0
   SK_STRING_ENCODING_UTF8 = 1
End Enum
```

Values

`SK_STRING_ENCODING_ANSI (0)`

Encode/decode using the code page for the current system locale.

`SK_STRING_ENCODING_UTF8 (1)`

Encode/decode using UTF-8.

SK_SystemIdentifierAlgorithm Enumeration

Enumeration for built-in system identifier algorithms.

Syntax

C/C++

```
typedef enum _SK_SystemIdentifierAlgorithm
{
   SK_SYSTEM_IDENTIFIER_ALGORITHM_NIC = 10,
   SK_SYSTEM_IDENTIFIER_ALGORITHM_COMPUTER_NAME = 20,
   SK_SYSTEM_IDENTIFIER_ALGORITHM_HARD_DISK_VOLUME_SERIAL = 30,
   SK_SYSTEM_IDENTIFIER_ALGORITHM_NETWORK_NAME = 40,
   SK_SYSTEM_IDENTIFIER_ALGORITHM_USER_NAME = 50
} SK_SystemIdentifierAlgorithm;
```

Visual Basic

```
Public Enum SK_SystemIdentifierAlgorithm
   SK_SYSTEM_IDENTIFIER_ALGORITHM_NIC = 10
   SK_SYSTEM_IDENTIFIER_ALGORITHM_COMPUTER_NAME = 20
   SK_SYSTEM_IDENTIFIER_ALGORITHM_HARD_DISK_VOLUME_SERIAL = 30
   SK_SYSTEM_IDENTIFIER_ALGORITHM_NETWORK_NAME = 40
   SK_SYSTEM_IDENTIFIER_ALGORITHM_USER_NAME = 50
End Enum
```

Values

`SK_SYSTEM_IDENTIFIER_ALGORITHM_NIC (10)`

Use for adding current identifiers for all relevant Network Interface Cards (NICs) using their MAC/physical addresses.

`SK_SYSTEM_IDENTIFIER_ALGORITHM_COMPUTER_NAME (20)`

Use for adding a current identifier for the computer's name.

`SK_SYSTEM_IDENTIFIER_ALGORITHM_HARD_DISK_VOLUME_SERIAL (30)`

Use for adding current identifiers for the volume format serials on all accessible partitions on all available hard drives which report themselves as "fixed." (Note that some removable drives such as some USB drives or hard drives placed in USB enclosures may still report themselves as being "fixed" drives.)

`SK_SYSTEM_IDENTIFIER_ALGORITHM_NETWORK_NAME (40)`

Use for adding identifiers a network share location that use being used to store the applications license file and/or network semaphore files.

`SK_SYSTEM_IDENTIFIER_ALGORITHM_USER_NAME (50)`

Use for adding an identifier for the username of the user which launched the calling process (the real user for Linux and Mac OS X -- the effective user cannot be retrieved presently) or thread (Windows).

SK_VirtualMachineTypes Enumeration

Enumeration for virtual machine types/hypervisors.

Syntax

C/C++

```
typedef enum _SK_VirtualMachineTypes
{
   SK_VIRTUAL_MACHINE_NONE = 0,
   SK_VIRTUAL_MACHINE_HYPERV = 1,
   SK_VIRTUAL_MACHINE_VIRTUALPC = 2,
   SK_VIRTUAL_MACHINE_VIRTUALBOX = 4,
   SK_VIRTUAL_MACHINE_VMWARE = 8,
   SK_VIRTUAL_MACHINE_PARALLELS = 16,
   SK_VIRTUAL_MACHINE_XENSERVER = 32
} SK_VirtualMachineTypes;
```

Visual Basic

```
Public Enum SK_VirtualMachineTypes
   SK_VIRTUAL_MACHINE_NONE = 0
   SK_VIRTUAL_MACHINE_HYPERV = 1
   SK_VIRTUAL_MACHINE_VIRTUALPC = 2
   SK_VIRTUAL_MACHINE_VIRTUALBOX = 4
   SK_VIRTUAL_MACHINE_VMWARE = 8
   SK_VIRTUAL_MACHINE_PARALLELS = 16
   SK_VIRTUAL_MACHINE_XENSERVER = 32
End Enum
```

Values

`SK_VIRTUAL_MACHINE_NONE (0)`

None

`SK_VIRTUAL_MACHINE_HYPERV (1)`

Microsoft Hyper V

`SK_VIRTUAL_MACHINE_VIRTUALPC (2)`

Microsoft Virtual PC

`SK_VIRTUAL_MACHINE_VIRTUALBOX (4)`

Oracle VirtualBox

`SK_VIRTUAL_MACHINE_VMWARE (8)`

VMWare

`SK_VIRTUAL_MACHINE_PARALLELS (16)`

Parallels

`SK_VIRTUAL_MACHINE_XENSERVER (32)`

Citrix XenServer

Functions

Name	Description
SK_ApiContextDispose	Disposes an **SK_ApiContext**, which clears it from memory and sets it's pointer to NULL (0).
SK_ApiContextInitialize	Initializes a new **API Context**, which may be used to open and manipulate a license file.
SK_ApiContextSetFieldInt	Sets an integer field in the **API Context**.
SK_ApiContextSetFieldString	Sets an string field in the **API Context**.
SK_ApiResultCodeToString	Generates a string containing a short, human-readable description of the result code.
SK_CallXmlWebService	Calls a SOLO Server XML web service method.
SK_DateTimeAddDays	Adds the specified number of days to the date-time provided.
SK_DateTimeAddMinutes	Adds the specified number of minutes to the date-time provided.
SK_DateTimeCompareStrings	Compares two date-time strings in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).
SK_DateTimeDaysRemaining	Determines the number of days left until a specified date is reached.
SK_DateTimeGetCurrentString	Retrieves the current date-time as a string from the system clock.
SK_DateTimeParseString	Parses a date-time string and returns the corresponding time_t value.
SK_DateTimeTimeRemaining	Determines the number of minutes left until a specified time is reached.
SK_DateTimeToFormattedString	Generates a date-time string with a specific format.
SK_DateTimeToString	Converts a time_t value into a corresponding date-time string in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).

SK_DateTimeValidateApi	Verifies that the operating system clock APIs have not been compromised. This prevents tools like "Time Stopper" or "RunAsDate" from being used to trick your protected applications.
SK_FileDelete	Deletes a file.
SK_FilePathIsRemote	Evaluates a path to determine whether it is at a network/remote location.
SK_FileReadAllText	Reads all text from a file to a string.
SK_FileWriteAllText	Writes all text from a string to a file.
SK_HttpConnectionTest	Performs an HTTP request to test connectivity with a server.
SK_HttpRequest	Performs an HTTP request.
SK_HttpUrlEncodeString	Encodes data using the **'application/x-www-form-urlencoded' encoding** method.
SK_PermissionsGrantControlToWorld	If possible, this function will give everyone access to read and write to the specified path.
SK_PLUS_LicenseAliasAdd	Adds a License File Alias to the list of aliases to be loaded when calling **SK_PLUS_LicenseFileLoad**.
SK_PLUS_LicenseAliasGetCount	Retrieves the number of aliases which were successfully saved.
SK_PLUS_LicenseAliasGetTotal	Retrieves the number of aliases configured for the license.
SK_PLUS_LicenseAliasGetValidatedCount	Retrieves the number of aliases which were successfully loaded and validated.
SK_PLUS_LicenseCreateNew	Creates a new License.
SK_PLUS_LicenseDispose	Disposes of the license in memory.
SK_PLUS_LicenseFileLoad	Loads a License from a file or Windows Registry key.
SK_PLUS_LicenseFileSave	Saves a License to a file or Windows Registry key.

SK_PLUS_LicenseGetDocumentString	Retrieves the raw contents of the License File.
SK_PLUS_LicenseGetXmlDocument	Retrieves the raw contents of the License File.
SK_PLUS_LicenseLoad	Loads a license from contents in memory.
SK_PLUS_LicenseProductOptionGetType	Gets the SOLO Server product option type for the current license.
SK_PLUS_LicenseProductOptionSetType	Sets the SOLO Server product option type for the current license.
SK_PLUS_NetworkSemaphoreCleanup	Cleans-up orphaned semaphore files from a directory, which can improve the performance of **SK_PLUS_NetworkSemaphoreOpen** (especially when all or nearly all network seats are in use). Calling this function is also necessary when using the **SK_FLAGS_NETWORK_SEMAPHORE_SKIPLOCKATTEMPT** flag with **SK_PLUS_NetworkSemaphoreOpen**.
SK_PLUS_NetworkSemaphoreDispose	Disposes of a network semaphore, unlocking and deleting the semaphore file.
SK_PLUS_NetworkSemaphoreOpen	Keeps track of concurrent users of an application on a network by using semaphore files in a shared network location.
SK_PLUS_NetworkSemaphoreStatistics	Obtains some basic statistics on network semaphore usage.
SK_PLUS_NetworkSemaphoreVerify	Verifies the process still has a valid lock on the semaphore file created with **SK_PLUS_NetworkSemaphoreOpen**.
SK_PLUS_NetworkSessionCheckin	Checks a Network Floating session back in.
SK_PLUS_NetworkSessionCheckout	Performs a session check-out for a given duration of time to enable offline use.
SK_PLUS_NetworkSessionClose	Closes a Network Floating Session.
SK_PLUS_NetworkSessionDispose	Dispose a Network Floating Session without closing it.
SK_PLUS_NetworkSessionGetCurrent	Retrieves the raw, encrypted contents of the

	current Network Session Certificate.
SK_PLUS_NetworkSessionLoad	Loads the Network Session Certificate from the file.
SK_PLUS_NetworkSessionOpen	Opens a Network Floating Session.
SK_PLUS_NetworkSessionPoll	Polls current Network Session Certificate to validate and maintain the status of the Network Floating Session.
SK_PLUS_SystemIdentifierAddCurrentIdentifier	Adds a SystemIdentifier element to the list of current system identifiers.
SK_PLUS_ SystemIdentifierAlgorithmAddCurrentIdentifiers	Adds SystemIdentifier elements to the list of current system identifiers using the specified, built-in algorithm.
SK_PLUS_SystemIdentifierCompare	Compares the current system's identifiers against the identifiers authorized in the license.
SK_PLUS_SystemIdentifierCurrentGetContents	Retrieves the contents of the current system's identifiers.
SK_PLUS_SystemIdentifierCurrentSetContents	Sets the contents of the current system's identifiers.
SK_PLUS4_GenerateUserCode1Value	Generates a Protection PLUS 4 compatible trigger code "User Code 1" (or Session Code) value, which may be used to make each activation attempt and each trigger code issued for a system unique.
SK_PLUS4_GenerateUserCode2Value	Generates a Protection PLUS 4 compatible trigger code "User Code 2" (or "Computer ID") value from the current system identifiers.
SK_PLUS4_GetLicenseStatusRequest	Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_GetLicenseStatus** and **SK_ GetLicenseStatusEx** functions.
SK_PLUS4_GetLicenseStatusResponse	Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_GetLicenseStatus** and **SK_GetLicenseStatusEx** functions.

SK_PLUS4_GetRegDataRequest	Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_GetRegData** and **SK_GetRegDataEx** functions.
SK_PLUS4_GetRegDataResponse	Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_GetRegData** and **SK_GetRegDataEx** functions.
SK_PLUS4_GetTcDataRequest	Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_GetTCData** and **SK_GetTCDataEx** functions.
SK_PLUS4_GetTcDataResponse	Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_GetTCData** and **SK_GetTCDataEx** functions.
SK_PLUS4_NDecrypt	Decrypts a number or RegKey2 value.
SK_PLUS4_NEncrypt	Encrypts a number or RegKey2 value.
SK_PLUS4_PostEvalDataRequest	Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_PostEvalData** and **SK_PostEvalDataEx** functions.
SK_PLUS4_PostEvalDataResponse	Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_PostRegData** or **SK_PostRegDataEx** functions.
SK_PLUS4_PostRegDataRequest	Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_PostRegData** > function.
SK_PLUS4_PostRegDataResponse	Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_PostRegData** function.
SK_PLUS4_ValidateTriggerCode	Determines whether or not a trigger code is valid.

SK_RegistryValueGet	Gets a Windows Registry key's string value.
SK_RegistryValueSet	Sets a Windows Registry key's string value.
SK_SessionCodeGenerate	Generates a randomized "session code" (similar to a cryptographic nonce).
SK_SOLO_ActivateInstallationGetRequest	Builds a request to send to the XmlActivationService web service's ActivateInstallationLicenseFile method in SOLO Server.
SK_SOLO_CheckInstallationStatusGetRequest	Builds a request to send to the XmlActivationService web service's CheckInstallationStatus method in SOLO Server.
SK_SOLO_DeactivateInstallationGetRequest	Builds a request to send to the XmlActivationService web service's DeactivateInstallation method in SOLO Server.
SK_SOLO_GetLicenseFileGetRequest	Builds a request to send to the XmlLicenseFileService web service's GetLicenseFile method in SOLO Server.
SK_SOLO_ManualRequestFileLoad	Loads and decrypts a manual response file.
SK_SOLO_ManualRequestFileSave	Saves a manual request file to the specified path.
SK_SOLO_NetworkSessionCheckinGetRequest	Builds a request to send to the XmlNetworkFloatingService web service's CheckinSession method in SOLO Server.
SK_SOLO_NetworkSessionCheckoutGetRequest	Builds a request to send to the XmlNetworkFloatingService web service's CheckoutSession method in SOLO Server.
SK_SOLO_NetworkSessionCloseGetRequest	Builds a request to send to the XmlNetworkFloatingService web service's CloseSession method in SOLO Server.
SK_SOLO_NetworkSessionOpenGetRequest	Builds a request to send to the XmlNetworkFloatingService web service's OpenSession method in SOLO Server.
SK_SOLO_NetworkSessionPollGetRequest	Builds a request to send to the XmlNetworkFloatingService web service's

	PollSession method in SOLO Server.
SK_StringConvertMultiByteStringToWideString	Converts a string with the specified encoding to a wide string.
SK_StringConvertWideStringToMultiByteString	Converts a wide string to a string using the specified encoding.
SK_StringCopyToBuffer	Copies a string to a fixed-size string buffer.
SK_StringCopyToBufferW	Copies a wide string to a fixed-size wide string buffer.
SK_StringDispose	Disposes a string, which clears it from memory and sets it's pointer to NULL (0).
SK_StringDisposeW	Disposes a wide string, which clears it from memory and sets it's pointer to NULL (0).
SK_StringGetLength	Gets the length of the specified string.
SK_StringGetLengthW	Gets the length of the specified wide string.
SK_StringToVersionNumbers	Parses the individual version number parts from a version string formatted like X.X.X.X, where each X is a positive integer value no longer than 5 digits in length.
SK_SystemRemoteSessionDetect	Detects whether or not the protected application is being run in a remote session via Terminal Services/Remote Desktop.
SK_SystemVirtualMachineDetect	Detects whether or not the protected application is being run in a virtual machine guest environment.
SK_XmlDocumentCreateFromString	Creates an XML document from a string representation.
SK_XmlDocumentDecryptRsa	Decrypts an XML document using the RSA Algorithm.
SK_XmlDocumentDispose	Disposes an XML document, which clears it from memory and sets it's pointer to NULL (0).
SK_XmlDocumentEncryptRsa	Encrypts an XML document using the RSA

	Algorithm.
SK_XmlDocumentGetDocumentString	Retrieves the raw contents of the XML document.
SK_XmlDocumentLoadFile	Loads an XML document from a file or registry value.
SK_XmlElementAddNew	Adds an element to a given XML document.
SK_XmlNodeGetDocument	Creates an new XML document from a sub-document in the specified XML document.
SK_XmlNodeGetValueDateTimeString	Retrieves a date-time string value from a given XML node.
SK_XmlNodeGetValueDouble	Retrieves a double floating point value from a given XML node.
SK_XmlNodeGetValueInt	Retrieves an integer value from a given XML node.
SK_XmlNodeGetValueString	Retrieves a string value from a given XML node.
SK_XmlNodeSetDocument	Imports an XML document under the node in the specified XML document.
SK_XmlNodeSetValueDateTimeString	Sets a node's text value to a given date-time string.
SK_XmlNodeSetValueDouble	Sets a node to a given double floating point value.
SK_XmlNodeSetValueInt	Sets a node to a given integer value.
SK_XmlNodeSetValueString	Sets a node's text value to a given string.

SK_ApiContextDispose Function

Disposes an **SK_ApiContext**, which clears it from memory and sets it's pointer to NULL (0).

Remarks

Important Note

The **SK_FLAGS_APICONTEXTDISPOSE_SHUTDOWN** flag must be passed the last time your application calls this function. This should be the last PLUSNative API function your application calls, and it should also be done from your application's main thread (especially when using **SK_FLAGS_APICONTEXTINITIALIZE_MAINTHREAD** with **SK_ApiContextInitialize**). This will shutdown the PLUSNative API and will free all memory.

Syntax

C/C++

```
SK_ResultCode SK_ApiContextDispose(int flags, SK_ApiContext *context);
```

Visual Basic

```
Declare Function SK_ApiContextDispose(ByVal flags As Long, ByRef context As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`context`

Reference/pointer to the API context to dispose.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ApiContextInitialize Function

Initializes a new **API Context**, which may be used to open and manipulate a license file.

Remarks

Important Note

The API context returned in the context parameter must be freed from memory! The **SK_ApiContextDispose** function is recommended for this purpose.

Important Note

If your application is multi-threaded, it is important that the first call to this function occurs on your application's main thread and specifies the **SK_FLAGS_APICONTEXTINITIALIZE_MAINTHREAD** flag. Failing to do so can cause global data for dependencies to be initialized on other threads, which may cause erroneous behavior and memory leaks. Calling this function with the SK_FLAGS_APICONTEXTINITIALIZE_MAINTHREAD flag more than once is not supported, and may also cause erroneous behavior.

Syntax

C/C++

```
SK_ResultCode SK_ApiContextInitialize(int flags, SK_BOOL licenseWritable, int prodId, int prodOptionId, const char *prodVersion, const char *envelope, SK_ApiContext *context);
```

Visual Basic

```
Declare Function SK_ApiContextInitialize(ByVal flags As Long, ByVal licenseWritable As Long, ByVal prodId As Long, ByVal prodOptionId As Long, ByVal prodVersion As String, ByVal envelope As String, ByRef context As Long) As Long
```

Arguments

`flags`

Any flags specific to this function-call.

`licenseWritable`

Specifies whether or not the API context will be used with writable license files.

`prodId`

The Product ID in SOLO Server from which licenses are created.

`prodOptionId`

(Optional - set to zero (0) to ignore.) The Product Option ID in SOLO Server from which licenses are created.

`prodVersion`

(Optional - set to an empty string or NULL (0) to ignore.) The Product Version number (must be formatted like X.X.X.X, where X is 5 digits or less).

`envelope`

The encryption key envelope from SOLO Server.

`context`

Reference/pointer to the new context which is initialized by this function. The context must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_ApiContextSetFieldInt Function

Sets an integer field in the **API Context**.

Syntax

C/C++

```
SK_ResultCode SK_ApiContextSetFieldInt(SK_ApiContext context, int flags, SK_ApiContext_
IntFields field, int value);
```

Visual Basic

```
Declare Function SK_ApiContextSetFieldInt(ByVal context As Long, ByVal flags As Long, ByVal
field As Long, ByVal value As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`field`

The unique Field ID for the field being set (see **SK_ApiContext_IntFields** for possible values).

`value`

The new value to store in the field.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ApiContextSetFieldString Function

Sets an string field in the **API Context**.

Syntax

C/C++

```
SK_ResultCode SK_ApiContextSetFieldString(SK_ApiContext context, int flags, SK_ApiContext_
StringFields field, const char *value);
```

Visual Basic

```
Declare Function SK_ApiContextSetFieldString(ByVal context As Long, ByVal flags As Long,
ByVal field As Long, ByVal value As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`field`

The unique Field ID for the field being set (see **SK_ApiContext_StringFields** for possible values).

`value`

The new value to store in the field.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_ApiResultCodeToString Function

Generates a string containing a short, human-readable description of the result code.

Remarks

Important Note

The string returned in the description parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_ApiResultCodeToString(int flags, SK_ResultCode resultCode, char **description);
```

Visual Basic

```
Declare Function SK_ApiResultCodeToString(ByVal flags As Long, ByVal resultCode As Long,
ByRef description As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`resultCode`

The result code - see **SK_ResultCode** for possible values and their descriptions.

`description`

Reference/pointer to a string pointer, which will point to the string which contains the result code description when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_CallXmlWebService Function

Calls a SOLO Server XML web service method.

Remarks

Important Note

The XML document returned in the response parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_CallXmlWebService(SK_ApiContext context, int flags, const char *url, SK_XmlDoc request, SK_XmlDoc *response, int *resultCode, int *statusCode);
```

Visual Basic

```
Declare Function SK_CallXmlWebService(ByVal context As Long, ByVal flags As Long, ByVal url As String, ByVal request As Long, ByRef response As Long, ByRef resultCode As Long, ByRef statusCode As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`url`

The SOLO Server XML web service URL to post to (not including http:// or https://). For Instant SOLO Server, constants such as **SK_CONST_WEBSERVICE_ACTIVATEINSTALLATION_URL** are provided.

`request`

An XML document containing the request content to send to the web service method.

`response`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle which contains the web service response when the call succeeds. The pointer must be initialized to NULL (0) prior to calling this function.

`resultCode`

Reference/pointer to an integer, which will point to the **result code** received from the web service.

`statusCode`

Reference/pointer to an integer, which will point to the HTTP response status code for this request.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_SESSION_VERIFICATION_FAILED	The requested session verification has failed.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_WEBSERVICE_CALL_FAILED	An unexpected failure occurred during an attempt to call a Web Service.
SK_ResultCode.SK_ERROR_WEBSERVICE_RETURNED_FAILURE	A call to a Web Service succeeded, but the functionality of the Web Service returned an indicator of failure.

SK_DateTimeAddDays Function

Adds the specified number of days to the date-time provided.

Syntax

C/C++

```
SK_ResultCode SK_DateTimeAddDays(int flags, char *dateTime, int days);
```

Visual Basic

```
Declare Function SK_DateTimeAddDays(ByVal flags As Long, ByVal dateTime As String, ByVal days
As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`dateTime`

The original date-time in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).

`days`

The number of days to add.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_DATETIME_CONVERSION_FAILED	A date-time conversion operation failed.

SK_DateTimeAddMinutes Function

Adds the specified number of minutes to the date-time provided.

Syntax

C/C++

```
SK_ResultCode SK_DateTimeAddMinutes(int flags, char *dateTime, int minutes);
```

Visual Basic

```
Declare Function SK_DateTimeAddMinutes(ByVal flags As Long, ByVal dateTime As String, ByVal
minutes As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`dateTime`

The original date-time in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).

`minutes`

The number of minutes to add.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_DATETIME_CONVERSION_FAILED	A date-time conversion operation failed.

SK_DateTimeCompareStrings Function

Compares two date-time strings in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).

Syntax

C/C++

```
SK_ResultCode SK_DateTimeCompareStrings(int flags, const char *dateTime1, const char *dateTime2, int *comparison);
```

Visual Basic

```
Declare Function SK_DateTimeCompareStrings(ByVal flags As Long, ByVal dateTime1 As String, ByVal dateTime2 As String, ByRef comparison As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`dateTime1`

A date-time string.

`dateTime2`

A date-time string.

`comparison`

Reference/pointer to an integer, which will point to a value of zero (0) if the dates are the same, a negative value if dateTime1 is less than dateTime2, or a positive value if dateTime1 is greater than dateTime2.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_DateTimeDaysRemaining Function

Determines the number of days left until a specified date is reached.

Syntax

C/C++

```
SK_ResultCode SK_DateTimeDaysRemaining(int flags, const char *dateTime, int *days);
```

Visual Basic

```
Declare Function SK_DateTimeDaysRemaining(ByVal flags As Long, ByVal dateTime As String,
ByRef days As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`dateTime`

The target date-time in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).

`days`

Reference/pointer to an integer, which will point to the number of days until the date in the dateTime parameter is reached when the call succeeds. (A negative value indicates that this date has already passed.)

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_DateTimeGetCurrentString Function

Retrieves the current date-time as a string from the system clock.

Remarks

Important Note

The string returned in the dateTime parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

The date-time string returned is in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ). If necessary, the **SK_DateTimeToFormattedString** function can be used to format the string as desired.

Syntax

C/C++

```
SK_ResultCode SK_DateTimeGetCurrentString(int flags, char **dateTime);
```

Visual Basic

```
Declare Function SK_DateTimeGetCurrentString(ByVal flags As Long, ByRef dateTime As String)
As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`dateTime`

Reference/pointer to a string pointer, which will point to the string which contains the current date-time value when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_DATETIME_CONVERSION_FAILED	A date-time conversion operation failed.

SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_DateTimeParseString Function

Parses a date-time string and returns the corresponding time_t value.

Syntax

C/C++

```
SK_ResultCode SK_DateTimeParseString(int flags, const char *dateTime, time_t *t);
```

Visual Basic

```
'Unsupported.
```

Arguments

`flags`

Any **flags** passed into this function-call.

`dateTime`

The date-time string to parse, in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).

`t`

Reference/pointer to a time_t, which will point to the parsed date-time value when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_DateTimeTimeRemaining Function

Determines the number of minutes left until a specified time is reached.

Syntax

C/C++

```
SK_ResultCode SK_DateTimeTimeRemaining(int flags, const char *dateTime, int *minutes);
```

Visual Basic

```
Declare Function SK_DateTimeTimeRemaining(ByVal flags As Long, ByVal dateTime As String,
ByRef minutes As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`dateTime`

The target date-time in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).

`minutes`

Reference/pointer to an integer, which will point to the number of minutes until the date-time in the dateTime parameter is reached when the call succeeds. (A negative value indicates that this date and time has already passed.)

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_DateTimeToFormattedString Function

Generates a date-time string with a specific format.

Remarks

Important Note

The string returned in the value parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_DateTimeToFormattedString(int flags, const char *dateTime, const char *format, char **value);
```

Visual Basic

```
Declare Function SK_DateTimeToFormattedString(ByVal flags As Long, ByVal dateTime As String, ByVal format As String, ByRef value As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`dateTime`

The original date-time string in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).

`format`

The format specifier string (this uses the same format specifiers as the standard C strftime function).

`value`

Reference/pointer to a string pointer, which will point to the string which contains the formatted date-time string when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_DATETIME_CONVERSION_FAILED	A date-time conversion operation failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_DateTimeToString Function

Converts a time_t value into a corresponding date-time string in ISO-8601 format (YYYY-MM-DDTHH:MM:SSZ).

Remarks

Important Note

The string returned in the dateTime parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_DateTimeToString(int flags, time_t t, char **dateTime);
```

Visual Basic

```
'Unsupported.
```

Arguments

`flags`

Any **flags** passed into this function-call.

`t`

The time_t value to convert.

`dateTime`

Reference/pointer to a string pointer, which will point to the string which contains the current date-time string when the call succeeds. The pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_DATETIME_CONVERSION_FAILED	A date-time conversion operation failed.

SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_DateTimeValidateApi Function

Verifies that the operating system clock APIs have not been compromised. This prevents tools like "Time Stopper" or "RunAsDate" from being used to trick your protected applications.

Remarks

Important Note

The accuracy of the reported time elapsed during this test may vary depending on the environment (particularly the processor). Consequently, making the test too short or too precise carries significant risk of a false positive. It is strongly recommended that you ensure all tests are at least 100 milliseconds in duration, and not required to be more than 90% accurate. (The default is 100 milliseconds with 75% accuracy.)

Syntax

C/C++

```
SK_ResultCode SK_DateTimeValidateApi(int flags, int testDuration, int thresholdPercent, int *change);
```

Visual Basic

```
Declare Function SK_DateTimeValidateApi(ByVal flags As Long, ByVal testDuration As Long, ByVal thresholdPercent As Long, ByRef change As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`testDuration`

The duration of time (in milliseconds) that will pass to complete this test. Pass 0 (zero) to use the default (100 milliseconds).

`thresholdPercent`

The percentage [1,100] of the test duration's time that must pass for the system's clock APIs to be considered valid. Pass 0 (zero) to use the default (75%).

`change`

The amount of time (in milliseconds) elapsed during the test. Pass NULL (0) to ignore this parameter.

Returns

SK_ResultCode.SK_ERROR_NONE is returned when validation passes. All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_SYSTEM_TIME_VERIFICATION_FAILED	Verification of system time has failed.
SK_ResultCode.SK_ERROR_SYSTEM_TIME_INVALID	System time is not valid.

SK_FileDelete Function

Deletes a file.

Syntax

C/C++

```
SK_ResultCode SK_FileDelete(int flags, const char *path);
```

Visual Basic

```
Declare Function SK_FileDelete(ByVal flags As Long, ByVal path As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`path`

The absolute path of the file to delete.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_COULD_NOT_DELETE_FILE	Could not delete file.

SK_FilePathIsRemote Function

Evaluates a path to determine whether it is at a network/remote location.

Remarks

Important Note

This function is only supported in Windows desktop environments.

Always verify the function returns a successful result before evaluating the value of pathIsRemote. When successful, the pathIsRemote value will contain TRUE (1) if the path resides on a network/remote resource, or FALSE (0) if it resides on the local device/system.

Syntax

C/C++

```
SK_ResultCode SK_FilePathIsRemote(int flags, const char *path, SK_BOOL *pathIsRemote);
```

Visual Basic

```
Declare Function SK_FilePathIsRemote(ByVal flags As Long, ByVal path As String, ByRef
pathIsRemote As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`path`

The path to a directory or volume to evaluate.

`pathIsRemote`

A pointer/reference to an BOOL (int) value, where the result is stored. Always evaluate this function's return code to check for errors before evaluating this value.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.

SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_UNSUPPORTED_OS	The current operating system is not supported by this feature or function.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_FileReadAllText Function

Reads all text from a file to a string.

Remarks

Important Note

The string returned in the text parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_FileReadAllText(int flags, const char *path, char **text);
```

Visual Basic

```
Declare Function SK_FileReadAllText(ByVal flags As Long, ByVal path As String, ByRef text As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`path`

The path to the file to read.

`text`

Reference/pointer to a string pointer, which will point to the string which contains the file's text content when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_FILE	An attempt to open a file failed.

SK_ResultCode.SK_ERROR_COULD_NOT_READ_FILE	An attempt to read a file failed.
SK_ResultCode.SK_ERROR_IO_OPERATION_FAILED	An attempt to perform an I/O operation failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_FileWriteAllText Function

Writes all text from a string to a file.

Syntax

C/C++

```
SK_ResultCode SK_FileWriteAllText(int flags, const char *path, const char *text);
```

Visual Basic

```
Declare Function SK_FileWriteAllText(ByVal flags As Long, ByVal path As String, ByVal text As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`path`

The path to the file which will be written.

`text`

The string of text to write to the file.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_FILE	An attempt to open a file failed.

SK_HttpConnectionTest Function

Performs an HTTP request to test connectivity with a server.

Remarks

Important Note

The string returned in the response parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

The SK_HttpConnectionTest function compliments the SK_HttpRequest function to provide a simplified means of testing connectivity with a server. Although testing connectivity with a server is simple in principle, it is not always so simple in practice. Complications are often encountered when the computer running the protected application is behind a proxy server. For example, proxy servers are sometimes configured with rules that are only considerate of how most web browsers behave, which can lead to issues that might be otherwise unusual to see in your applications.

When the searchString parameter is NULL (0) or an empty string (""), SK_HttpConnectionTest will search for `<div id="HttpConnectionTest">[FQDN]</div>` by default (where [FQDN] is the server's fully-qualified domain name, such as secure.softwarekey.com). This default search string is output by SOLO Server's webservices/ping.aspx page (for Instant SOLO Server users, the complete URL is secure.softwarekey.com/solo/webservices/ping.aspx).

Syntax

C/C++

```
SK_ResultCode SK_HttpConnectionTest(SK_ApiContext context, int flags, const char *url, const char *searchString, char **response, int *statusCode);
```

Visual Basic

```
Declare Function SK_HttpConnectionTest(ByVal context As Long, ByVal flags As Long, ByVal url As String, ByVal searchString As String, ByRef response As String, ByRef statusCode As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`url`

The URL for which the HTTP request will be performed. The value provided must not begin with http:// or https://, as the protocol used is determined by the context's flags or the flags argument.

`searchString`

The string that must be present in the server's response to indicate there is valid connectivity. Pass NULL (0) or an empty string ("") to use the server's fully-qualified domain name (FQDN) in a div tag (see remarks) as the search string.

`response`

Reference/pointer to a string pointer, which will point to the string which contains the response returned from the server when the call succeeds. Pass NULL (0) to ignore this parameter.

`statusCode`

Reference/pointer to an integer, which will point to the HTTP response status code for this request. Pass NULL (0) to ignore this parameter.

Returns

If the server responds with the expected output, **SK_ResultCode.SK_ERROR_NONE** is returned. If the response does not contain the expected output, **SK_ResultCode.SK_ERROR_INVALID_DATA** is returned. All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_WEBSERVICE_CALL_FAILED	An unexpected failure occurred during an attempt to call a Web Service.
SK_ResultCode.SK_ERROR_HTTP_INITIALIZATION_FAILED	The HTTP client failed to initialize.
SK_ResultCode.SK_ERROR_HTTP_CONNECTION_FAILED	The server could not be reached. Verify that the computer is connected to the Internet, and that the firewall/proxy is set-up properly.
SK_ResultCode.SK_ERROR_HTTP_COULD_NOT_RESOLVE_HOST	The server could not be located. Verify that the computer is connected to the Internet, and that the firewall/proxy is set-up properly.
SK_ResultCode.SK_ERROR_SSL_FAILED	The HTTPS request failed due to an SSL related error.

SK_ResultCode.SK_ERROR_SSL_CERTIFICATE_UNAVAILABLE	The client certificate for SSL communication could not be found.

SK_HttpRequest Function

Performs an HTTP request.

Remarks

Important Note

The string returned in the response parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_HttpRequest(SK_ApiContext context, int flags, const char *url, const char *postString, char **response, int *statusCode);
```

Visual Basic

```
Declare Function SK_HttpRequest(ByVal context As Long, ByVal flags As Long, ByVal url As String, ByVal postString As String, ByRef response As String, ByRef statusCode As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`url`

The URL for which the HTTP request will be performed. The value provided must not begin with http:// or https://, as the protocol used is determined by the context's flags or the flags argument.

`postString`

For HTTP POST, pass the URL-encoded post data string. For HTTP GET, pass NULL or an empty string.

`response`

Reference/pointer to a string pointer, which will point to the string which contains the response returned from the server when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

`statusCode`

Reference/pointer to an integer, which will point to the HTTP response status code for this request.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_WEBSERVICE_CALL_FAILED	An unexpected failure occurred during an attempt to call a Web Service.
SK_ResultCode.SK_ERROR_HTTP_INITIALIZATION_FAILED	The HTTP client failed to initialize.
SK_ResultCode.SK_ERROR_HTTP_CONNECTION_FAILED	The server could not be reached. Verify that the computer is connected to the Internet, and that the firewall/proxy is set-up properly.
SK_ResultCode.SK_ERROR_HTTP_COULD_NOT_RESOLVE_HOST	The server could not be located. Verify that the computer is connected to the Internet, and that the firewall/proxy is set-up properly.
SK_ResultCode.SK_ERROR_SSL_FAILED	The HTTPS request failed due to an SSL related error.
SK_ResultCode.SK_ERROR_SSL_CERTIFICATE_UNAVAILABLE	The client certificate for SSL communication could not be found.

SK_HttpUrlEncodeString Function

Encodes data using the **'application/x-www-form-urlencoded' encoding** method.

Remarks

Important Note

The string returned in the output parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_HttpUrlEncodeString(int flags, char *input, const char *label, const char *separator, char **output);
```

Visual Basic

```
Declare Function SK_HttpUrlEncodeString(ByVal flags As Long, ByVal input As String, ByVal label As String, ByVal separator As String, ByRef output As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`input`

The data that needs to be encoded.

`label`

The field name/label.

`separator`

The separator to use between fields (typically &).

`output`

Reference/pointer to a string pointer, which will point to the string which contains the encoded string data when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PermissionsGrantControlToWorld Function

If possible, this function will give everyone access to read and write to the specified path.

Remarks

On most operating systems (including Windows, OS X, and Linux), the execute permission flag/bit determines whether or not traversal is allowed when applied to a directory.

Using the **SK_FLAGS_PERMISSIONS_FOLDERINHERIT** flag and/or a **Windows Registry** path type is only supported in Windows. Therefore, using these in any other operating system will cause the function to fail and return **SK_ResultCode.SK_ERROR_UNSUPPORTED_OS**.

On non-Windows, POSIX compatible operating systems (like OS X and Linux), the permissions are set without any regard to any prior permissions. So for example, if everyone is allowed to execute a file before calling this function, calling this function against that file without the **SK_FLAGS_PERMISSIONS_ALLOWEXECUTE** flag will result in nobody having access to execute that file. Additionally, permissions are never inherited or set recursively when applied to a directory.

Syntax

C/C++

```
SK_ResultCode SK_PermissionsGrantControlToWorld(int flags, SK_PathType type, const char *path);
```

Visual Basic

```
Declare Function SK_PermissionsGrantControlToWorld(ByVal flags As Long, ByVal type As Long, ByVal path As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`type`

The type of path (see **SK_PathType**).

`path`

The absolute path to the file or Windows Registry key location.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_REGISTRY_KEY	An attempt to open a registry key failed.
SK_ResultCode.SK_ERROR_COULD_NOT_SET_PERMISSIONS	An attempt to set a file or registry key's permissions failed.
SK_ResultCode.SK_ERROR_UNSUPPORTED_OS	The current operating system is not supported by this feature or function.

SK_PLUS_LicenseAliasAdd Function

Adds a License File Alias to the list of aliases to be loaded when calling **SK_PLUS_LicenseFileLoad**.

Remarks

Important Note

This function calls **SK_PermissionsGrantControlToWorld** to help ensure all users have access.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseAliasAdd(SK_ApiContext context, int flags, const char *path);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseAliasAdd(ByVal context As Long, ByVal flags As Long, ByVal
path As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`path`

The absolute path to the License file or Windows Registry Key location.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS_LicenseAliasGetCount Function

Retrieves the number of aliases which were successfully saved.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseAliasGetCount(SK_ApiContext context, int flags, int *count);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseAliasGetCount(ByVal context As Long, ByVal flags As Long,
ByRef count As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`count`

Reference/pointer to an integer, which will point to the number of aliases which were successfully saved.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_LicenseAliasGetTotal Function

Retrieves the number of aliases configured for the license.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseAliasGetTotal(SK_ApiContext context, int flags, int *count);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseAliasGetTotal(ByVal context As Long, ByVal flags As Long,
ByRef count As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`count`

Reference/pointer to an integer, which will point to the number of aliases configured for the license when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_LicenseAliasGetValidatedCount Function

Retrieves the number of aliases which were successfully loaded and validated.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseAliasGetValidatedCount(SK_ApiContext context, int flags, int *count);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseAliasGetValidatedCount(ByVal context As Long, ByVal flags As Long, ByRef count As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`count`

Reference/pointer to an integer, which will point to the number of aliases which were successfully loaded and validated.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_LicenseCreateNew Function

Creates a new License.

Remarks

Important Note

The XML document returned in the license parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseCreateNew(SK_ApiContext context, int flags, int daysTillExpiration, SK_XmlDoc *license);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseCreateNew(ByVal context As Long, ByVal flags As Long, ByVal daysTillExpiration As Long, ByRef license As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`daysTillExpiration`

The number of days (from the current date) until the license expires.

`license`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle which contains the License File contents when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS_LicenseDispose Function

Disposes of the license in memory.

Remarks

Calling this function directly is generally unnecessary. The primary use for this is for reloading license data from memory; however, functions such as **SK_PLUS_LicenseLoad**, **SK_PLUS_LicenseFileLoad**, and **SK_ApiContextDispose** automatically dispose of the content as necessary.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseDispose(SK_ApiContext context, int flags);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseDispose(ByVal context As Long, ByVal flags As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_LicenseFileLoad Function

Loads a License from a file or Windows Registry key.

Remarks

Important Note

This function calls **SK_PermissionsGrantControlToWorld** to help ensure all users have access.

Important Note

Each **API context** can only open a single license file. If your application uses multiple license files, it should use a separate API context for each license file.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseFileLoad(SK_ApiContext context, int flags, const char *path);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseFileLoad(ByVal context As Long, ByVal flags As Long, ByVal
path As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`path`

The absolute path to the License file or Windows Registry Key location.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_DECRYPTION_FAILED	The requested decryption operation has

	failed.
SK_ResultCode.SK_ERROR_VERIFICATION_FAILED	The requested verification operation has failed.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_COULD_NOT_LOAD_LICENSE	License could not be loaded.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_FILE	An attempt to open a file failed.
SK_ResultCode.SK_ERROR_COULD_NOT_READ_FILE	An attempt to read a file failed.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_ REGISTRY_KEY	An attempt to open a registry key failed.
SK_ResultCode.SK_ERROR_COULD_NOT_READ_ REGISTRY_KEY	An attempt to read a registry key value failed.
SK_ResultCode.SK_ERROR_IO_OPERATION_FAILED	An attempt to perform an I/O operation failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS_LicenseFileSave Function

Saves a License to a file or Windows Registry key.

Remarks

Important Note

This function calls **SK_PermissionsGrantControlToWorld** to help ensure all users have access.

Important Note

When implementing shared network licenses, it is strongly recommended to re-load the license file (using **SK_PLUS_LicenseFileLoad** or **SK_PLUS_LicenseLoad**), update the data as needed, and then save the license file. Although this will increase network traffic, it will help avoid synchronization issues when the same license file is shared by multiple users.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseFileSave(SK_ApiContext context, int flags, const char *path, SK_XmlDoc license);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseFileSave(ByVal context As Long, ByVal flags As Long, ByVal path As String, ByVal license As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`path`

The absolute path to the License file or Windows Registry Key location.

`license`

An XML document containing the contents of the license. **SK_PLUS_LicenseGetXmlDocument** may be used to retrieve these contents.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_ENCRYPTION_FAILED	The requested encryption operation has failed.
SK_ResultCode.SK_ERROR_SIGNING_FAILED	The requested signing operation has failed.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_FILE	An attempt to open a file failed.
SK_ResultCode.SK_ERROR_COULD_NOT_WRITE_FILE	An attempt to write a file failed.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_REGISTRY_KEY	An attempt to open a registry key failed.
SK_ResultCode.SK_ERROR_COULD_NOT_WRITE_REGISTRY_KEY	An attempt to write a registry key value failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS_LicenseGetDocumentString Function

Retrieves the raw contents of the License File.

Remarks

Important Note

The string returned in the content parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseGetDocumentString(SK_ApiContext context, int flags, char **content);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseGetDocumentString(ByVal context As Long, ByVal flags As Long, ByRef content As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`content`

Reference/pointer to a string pointer, which will point to the string which contains the License File contents when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.

SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_LicenseGetXmlDocument Function

Retrieves the raw contents of the License File.

Remarks

Important Note

The XML document returned in the license parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseGetXmlDocument(SK_ApiContext context, int flags, SK_XmlDoc *license);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseGetXmlDocument(ByVal context As Long, ByVal flags As Long, ByRef license As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`license`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle which contains the License File contents when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.

SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_LicenseLoad Function

Loads a license from contents in memory.

Remarks

Important Note

Each **API context** can only open a single license file. If your application uses multiple license files, it should use a separate API context for each license file.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseLoad(SK_ApiContext context, int flags, SK_XmlDoc license);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseLoad(ByVal context As Long, ByVal flags As Long, ByVal
license As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`license`

An XML document with the contents of the license to load.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_DECRYPTION_FAILED	The requested decryption operation has failed.
SK_ResultCode.SK_ERROR_VERIFICATION_FAILED	The requested verification operation has failed.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.

SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS_LicenseProductOptionGetType Function

Gets the SOLO Server product option type for the current license.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseProductOptionGetType(SK_ApiContext context, int flags, SK_ProductOptionType *type);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseProductOptionGetType(ByVal context As Long, ByVal flags As Long, ByRef type As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`type`

Reference/pointer to a **SK_ProductOptionType** variable, which will point to the product option type value when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_LicenseProductOptionSetType Function

Sets the SOLO Server product option type for the current license.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_LicenseProductOptionSetType(SK_ApiContext context, int flags, SK_ProductOptionType type);
```

Visual Basic

```
Declare Function SK_PLUS_LicenseProductOptionSetType(ByVal context As Long, ByVal flags As Long, ByVal type As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`type`

The **SK_ProductOptionType** value to set.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_NetworkSemaphoreCleanup Function

Cleans-up orphaned semaphore files from a directory, which can improve the performance of **SK_PLUS_NetworkSemaphoreOpen** (especially when all or nearly all network seats are in use). Calling this function is also necessary when using the **SK_FLAGS_NETWORK_SEMAPHORE_SKIPLOCKATTEMPT** flag with **SK_PLUS_NetworkSemaphoreOpen**.

Remarks

The semaphorePath and semaphorePrefix parameters are required unless you have a semaphore open from a prior, successful call to **SK_PLUS_NetworkSemaphoreOpen**. If you provide a null or empty string after opening a semaphore, this function will automatically use the same semaphore path and prefix used to open the semaphore.

By default, this function creates a thread so the application can attempt to delete any orphaned semaphore files without experiencing any visible delay in the protected application. When using this functionality, it is important for you to test thoroughly to ensure your application is not adversely affected by the thread that is created. (For example, there are some cases where creating threads can be problematic, such as when certain or custom threading models are used. There are other cases with limitations to consider with threading, such as during DLL initialization.) The thread created by this function is automatically terminated when: this function is called again, the **SK_PLUS_NetworkSemaphoreDispose** function is called, or the **SK_ApiContextDispose** function is called.

Important Note

Having your application periodically attempt to delete all orphaned semaphore files can cause network congestion (especially with larger-scale deployments), and this needs to be considered before deploying your applications to production environments. For this reason, this function enforces a minimum of 600 seconds (10 minutes) in the delay parameter (when not specifying zero).

If you have concerns about threading or network congestion, you should consider the following options for your application:

1. You can specify the **SK_FLAGS_NETWORK_SEMAPHORE_NOCLEANUPTHREAD** flag when calling this function. This causes the function to only attempt to delete orphaned semaphores once, and it also forces it to run synchronously (without creating any threads). This enables you to create your own implementation, which can be used to have full control over how often the application attempts to delete orphaned files, and gives you a way to run this functionality from threads your application creates and manages itself.
2. If you have concerns about orphaned semaphore files causing delays, but also have concerns about network congestion from deleting files too frequently, one option to consider is to use this function in your application with a delay of zero. This causes the function to only attempt deleting orphaned semaphore files once, which can help limit the potential for network congestion. However, with this approach, it is still possible to cause some extra congestion if a large number of users launch your application around the same time (e.g. at the start of a work day, or after lunch hours end).
3. If you have concerns about orphaned semaphore files causing delays, but also have concerns about network congestion from deleting files too frequently, another option to consider is to use this function in a separate application or service that runs on the server hosting the share. This would allow you to have a single process periodically deleting files, which allows you to avoid the concerns with congestion that can occur from having all running instances of your application attempting to delete orphaned semaphore files.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSemaphoreCleanup(SK_ApiContext context, int flags, const char
*semaphorePath, const char *semaphorePrefix, int delay);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSemaphoreCleanup(ByVal context As Long, ByVal flags As Long,
ByVal semaphorePath As String, ByVal semaphorePrefix As String, ByVal delay As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`semaphorePath`

The absolute path to the directory or network share where semaphore files will be created and locked. When using a network share, a UNC-formatted path is strongly recommended.

`semaphorePrefix`

The prefix to use for semaphore file names.

`delay`

After deleting files, this is amount of time (in seconds) to wait before trying to delete orphaned semaphore files again. The value specified must be greater than or equal to 600 seconds (10 minutes), or equal to zero. Specifying zero (0) causes this function to only delete orphaned semaphore files one time. This must have a value of zero (0) when specifying the **SK_FLAGS_NETWORK_SEMAPHORE_NOCLEANUPTHREAD** flag.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_NETWORK_SEMAPHORE_INVALID_PATH	The network path is not valid or is unavailable.
SK_ResultCode.SK_ERROR_UNSUPPORTED_OS	The current operating system is not supported by this feature or function.

SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS_NetworkSemaphoreDispose Function

Disposes of a network semaphore, unlocking and deleting the semaphore file.

Remarks

This function is automatically called by **SK_PLUS_NetworkSemaphoreOpen** to clear any previously opened semaphore on the given context. This is also automatically called by **SK_ApiContextDispose** to clear any opened semaphore on the context which is being disposed, which prevents memory leaks and helps avoid leaving abandoned semaphore files.

When called directly from your application after **SK_PLUS_NetworkSemaphoreCleanup** was called without the **SK_FLAGS_NETWORK_SEMAPHORE_NOCLEANUPTHREAD** flag, the thread created for deleting orphaned semaphore files is terminated.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSemaphoreDispose(SK_ApiContext context, int flags);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSemaphoreDispose(ByVal context As Long, ByVal flags As Long)
As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_NetworkSemaphoreOpen Function

Keeps track of concurrent users of an application on a network by using semaphore files in a shared network location.

Remarks

A Network Semaphore File is a file, which contains no useful data, stored on a common network directory used to assist in enforcing a Network Floating License in an application. One file is created and locked exclusively for each active workstation or instance of your protected application. Other LAN users attempting to run the application will not be able to use any semaphore files that are locked, which is how this can enforce license compliance which limits the number of network "seats."

Any users who will use an application using this type of licensing will need access to read, write, modify, and delete the semaphore files.

Important Note

The absolute path to the semaphore files (including the name of each semaphore file) cannot exceed the **maximum path limitation** (260 characters). If the path does exceed the length limitation, this function may return **SK_ResultCode.SK_ERROR_NETWORK_LICENSE_FULL** as a result of not being able to create and lock any semaphores.

This functionality is supported on environments where semaphore files are hosted on a Windows (SMB/CIFS) share, which functions best on a Local Area Network (LAN). While this may be used in a Wide Area Network (WAN) environment (which includes scenarios where a user access a given "LAN" over a Virtual Private Network [VPN] connection), it is strongly advised testing is done before deploying to any given WAN environment. This is because the performance of this feature is centered around that of the Windows (SMB/CIFS) shares being used to host the semaphore files (and WAN/VPN environments can see much slower performance than a typical LAN configuration).

Important Note

The NetworkSemaphore class is only supported when the system hosting the Windows (SMB/CIFS) share that houses the semaphore files and the clients which access the share all run Windows Vista/Windows Server 2008 or later.

Important Note

The computer hosting the Windows (SMB/CIFS) share (the server) that houses the semaphore files is ultimately responsible for handling when a file is unlocked. In most cases, applications which are terminated gracefully will unlock and delete their semaphore files. If the application crashes, however, the server will eventually unlock the semaphore file, but it will not delete the file. Having many of these files present can lead to some performance degradation, as each of these orphaned files can take a while to try to lock or delete. The **SK_PLUS_NetworkSemaphoreCleanup** function can help avoid or limit this kind of performance degradation.

With very large and/or WAN deployments, the performance degradation described above can often be avoided or limited by using the **SK_FLAGS_NETWORK_SEMAPHORE_SKIPLOCKATTEMPT** flag. This flag prevents this function from attempting to lock existing files (which it does to check if perhaps one of the existing files has been orphaned). While this flag can improve performance, keep in mind that using the flag also means you are required to call **SK_PLUS_NetworkSemaphoreCleanup** to delete orphaned semaphore files. If your application uses this flag without calling **SK_PLUS_NetworkSemaphoreCleanup**, it is likely that the presence of orphaned files will cause this function to prematurely return **SK_ResultCode.SK_ERROR_NETWORK_LICENSE_FULL** (meaning no seats are available).

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSemaphoreOpen(SK_ApiContext context, int flags, int
seatsAllowed, const char *seatsAllowedXPath, const char *semaphorePath, const char *sem-
aphorePrefix, const char *optionalIdentifier, char **filePath);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSemaphoreOpen(ByVal context As Long, ByVal flags As Long,
ByVal seatsAllowed As Long, ByVal seatsAllowedXPath As String, ByVal semaphorePath As String,
ByVal semaphorePrefix As String, ByVal optionalIdentifier As String, ByRef filePath As
String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`seatsAllowed`

The maximum number of seats allowed. When seatsAllowedXPath is not null or empty, the specified license file value overrides this value.

`seatsAllowedXPath`

An XPath that specifies the license file property/field, which overrides the seatsAllowed parameter. If this parameter receives a null or empty string, the seatsAllowed value is used.

`semaphorePath`

The absolute path to the directory or network share where semaphore files will be created and locked. When using a network share, a UNC-formatted path is strongly recommended.

`semaphorePrefix`

The prefix to use for semaphore file names.

`optionalIdentifier`

Reserved for future use. Pass a null or empty string.

`filePath`

The path to the semaphore file which was created and locked.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_XML_NODE_MISSING	The requested XML node could not be found.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_NETWORK_SEMAPHORE_INVALID_PATH	The network path is not valid or is unavailable.
SK_ResultCode.SK_ERROR_NETWORK_LICENSE_FULL	The number of allowed concurrent users has been reached.
SK_ResultCode.SK_ERROR_NETWORK_SEMAPHORE_LOCK_FAILED	Failed to create or lock the network semaphore.
SK_ResultCode.SK_ERROR_UNSUPPORTED_OS	The current operating system is not supported by this feature or function.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS_NetworkSemaphoreStatistics Function

Obtains some basic statistics on network semaphore usage.

Remarks

Important Note

It is important to call **SK_PLUS_NetworkSemaphoreCleanup** before calling this function to obtain accurate values that do not include orphaned files.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSemaphoreStatistics(SK_ApiContext context, int flags, const char *semaphorePath, const char *semaphorePrefix, int *seatsAllowed, int *seatsAllocated, int *seatsAvailable);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSemaphoreStatistics(ByVal context As Long, ByVal flags As Long, ByVal semaphorePath As String, ByVal semaphorePrefix As String, ByRef seatsAllowed As Long, ByRef seatsAllocated As Long, ByRef seatsAvailable As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`semaphorePath`

The absolute path to the directory or network share where semaphore files will be created and locked. When using a network share, a UNC-formatted path is strongly recommended.

`semaphorePrefix`

The prefix to use for semaphore file names.

`seatsAllowed`

When given a pointer to a 32 bit integer, its value will be set to the number of seats allowed on the session **opened** with the provided context. If no session was opened on the provided context, this will always receive a value of zero.

`seatsAllocated`

When given a pointer to a 32 bit integer, its value will be set to the number of seats presently allocated (or in-use). Note that it is important to use **SK_PLUS_NetworkSemaphoreCleanup** before calling this function to obtain an accurate count that does not include orphaned files.

`seatsAvailable`

When given a pointer to a 32 bit integer, its value will be set to the number of seats available (or not in-use). Note that it is important to use **SK_PLUS_NetworkSemaphoreCleanup** before calling this function to obtain an accurate count that does not include orphaned files.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_NETWORK_SEMAPHORE_INVALID_PATH	The network path is not valid or is unavailable.
SK_ResultCode.SK_ERROR_NETWORK_LICENSE_FULL	The number of allowed concurrent users has been reached.
SK_ResultCode.SK_ERROR_UNSUPPORTED_OS	The current operating system is not supported by this feature or function.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS_NetworkSemaphoreVerify Function

Verifies the process still has a valid lock on the semaphore file created with **SK_PLUS_NetworkSemaphoreOpen**.

Remarks

A lock on a semaphore file created with **SK_PLUS_NetworkSemaphoreOpen** can be lost any time the semaphore path is no longer accessible. This can occur when network connectivity is lost or interrupted, which is especially possible when a WAN (or VPN) connection is used to access the semaphore path. Once the lock is lost, another user or process can acquire a lock on the orphaned semaphore file, which is why it is important for your application to periodically use this function to verify that its lock on the semaphore file is still valid. This allows you to prevent your application from performing certain functions when it loses its network seat; however, you should also keep in mind that its important to avoid causing loss of your users' data or work in the process.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSemaphoreVerify(SK_ApiContext context, int flags);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSemaphoreVerify(ByVal context As Long, ByVal flags As Long)
As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_NETWORK_SEMAPHORE_LOCK_FAILED	Failed to create or lock the network semaphore.
SK_ResultCode.SK_ERROR_UNSUPPORTED_OS	The current operating system is not supported by this feature or function.

SK_PLUS_NetworkSessionCheckin Function

Important

This function is preliminary, and is subject to change.

Checks a Network Floating session back in.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSessionCheckin(SK_ApiContext context, int flags, SK_XmlDoc certificate);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSessionCheckin(ByVal context As Long, ByVal flags As Long, ByVal certificate As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`certificate`

An XML document containing the encrypted certificate. **SK_SOLO_NetworkSessionCheckinGetRequest** in combination with **SK_CallXmlWebService** should be used to retrieve this certificate.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_DECRYPTION_FAILED	The requested decryption operation has failed.
SK_ResultCode.SK_ERROR_VERIFICATION_FAILED	The requested verification operation has failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.
SK_ResultCode.SK_ERROR_COULD_NOT_DELETE_FILE	Could not delete file.
SK_ResultCode.SK_ERROR_LICENSE_SYSTEM_ IDENTIFIERS_DONT_MATCH	License system identifiers do not match.
SK_ResultCode.SK_ERROR_LICENSE_NOT_EFFECTIVE_YET	License is not yet effective.
SK_ResultCode.SK_ERROR_LICENSE_EXPIRED	License has expired.

SK_PLUS_NetworkSessionCheckout Function

Important

This function is preliminary, and is subject to change.

Performs a session check-out for a given duration of time to enable offline use.

Remarks

The loaded certificate file will automatically be locked to the process. If it is not desired to lock the file use the **SK_FLAGS_NETWORKSESSION_SKIPLOCKATTEMPT** flag.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSessionCheckout(SK_ApiContext context, int flags, SK_XmlDoc certificate);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSessionCheckout(ByVal context As Long, ByVal flags As Long, ByVal certificate As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`certificate`

An XML document containing the encrypted certificate. **SK_SOLO_NetworkSessionCheckoutGetRequest** in combination with **SK_CallXmlWebService** should be used to retrieve this certificate.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an

	error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_DECRYPTION_FAILED	The requested decryption operation has failed.
SK_ResultCode.SK_ERROR_VERIFICATION_FAILED	The requested verification operation has failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_FILE	An attempt to open a file failed.
SK_ResultCode.SK_ERROR_LICENSE_SYSTEM_ IDENTIFIERS_DONT_MATCH	License system identifiers do not match.
SK_ResultCode.SK_ERROR_LICENSE_NOT_EFFECTIVE_YET	License is not yet effective.
SK_ResultCode.SK_ERROR_LICENSE_EXPIRED	License has expired.

SK_PLUS_NetworkSessionClose Function

Important

This function is preliminary, and is subject to change.

Closes a Network Floating Session.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSessionClose(SK_ApiContext context, int flags);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSessionClose(ByVal context As Long, ByVal flags As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_NETWORK_CERTIFICATE_REQUIRED	A valid network session certificate is required, but is not present.

SK_PLUS_NetworkSessionDispose Function

Important

This function is preliminary, and is subject to change.

Dispose a Network Floating Session without closing it.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSessionDispose(SK_ApiContext context, int flags);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSessionDispose(ByVal context As Long, ByVal flags As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_NETWORK_CERTIFICATE_REQUIRED	A valid network session certificate is required, but is not present.

SK_PLUS_NetworkSessionGetCurrent Function

Important

This function is preliminary, and is subject to change.

Retrieves the raw, encrypted contents of the current Network Session Certificate.

Remarks

Important Note

The XML document returned in the license parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSessionGetCurrent(SK_ApiContext context, int flags, SK_XmlDoc
*certificate);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSessionGetCurrent(ByVal context As Long, ByVal flags As Long,
ByRef certificate As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`certificate`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle which contains the Network Session Certificate contents when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_NetworkSessionLoad Function

Important

This function is preliminary, and is subject to change.

Loads the Network Session Certificate from the file.

Remarks

The loaded certificate file will automatically be locked to the process. If it is not desired to lock the file use the **SK_FLAGS_NETWORKSESSION_SKIPLOCKATTEMPT** flag.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSessionLoad(SK_ApiContext context, int flags, const char *path);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSessionLoad(ByVal context As Long, ByVal flags As Long, ByVal
path As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`path`

The absolute path to the Network Session Certificate file.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.

SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_DECRYPTION_FAILED	The requested decryption operation has failed.
SK_ResultCode.SK_ERROR_VERIFICATION_FAILED	The requested verification operation has failed.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_FILE	An attempt to open a file failed.
SK_ResultCode.SK_ERROR_COULD_NOT_READ_FILE	An attempt to read a file failed.
SK_ResultCode.SK_ERROR_IO_OPERATION_FAILED	An attempt to perform an I/O operation failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.
SK_ResultCode.SK_ERROR_LICENSE_SYSTEM_IDENTIFIERS_DONT_MATCH	License system identifiers do not match.
SK_ResultCode.SK_ERROR_LICENSE_NOT_EFFECTIVE_YET	License is not yet effective.
SK_ResultCode.SK_ERROR_LICENSE_EXPIRED	License has expired.

SK_PLUS_NetworkSessionOpen Function

Important

This function is preliminary, and is subject to change.

Opens a Network Floating Session.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSessionOpen(SK_ApiContext context, int flags, SK_XmlDoc cer-
tificate);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSessionOpen(ByVal context As Long, ByVal flags As Long, ByVal
certificate As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`certificate`

An XML document containing the encrypted certificate. **SK_SOLO_NetworkSessionOpenGetRequest** in combination **SK_CallXmlWebService** should be used to retrieve this certificate.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_DECRYPTION_FAILED	The requested decryption operation has failed.
SK_ResultCode.SK_ERROR_VERIFICATION_FAILED	The requested verification operation has failed.
SK_ResultCode.SK_ERROR_LICENSE_SYSTEM_IDENTIFIERS_DONT_MATCH	License system identifiers do not match.
SK_ResultCode.SK_ERROR_LICENSE_NOT_EFFECTIVE_YET	License is not yet effective.
SK_ResultCode.SK_ERROR_LICENSE_EXPIRED	License has expired.

SK_PLUS_NetworkSessionPoll Function

Important

This function is preliminary, and is subject to change.

Polls current Network Session Certificate to validate and maintain the status of the Network Floating Session.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_NetworkSessionPoll(SK_ApiContext context, int flags, SK_XmlDoc certificate);
```

Visual Basic

```
Declare Function SK_PLUS_NetworkSessionPoll(ByVal context As Long, ByVal flags As Long, ByVal certificate As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`certificate`

An XML document containing the encrypted certificate. **SK_SOLO_NetworkSessionPollGetRequest** in combination with **SK_CallXmlWebService** should be used to retrieve this certificate.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_DECRYPTION_FAILED	The requested decryption operation has failed.
SK_ResultCode.SK_ERROR_VERIFICATION_FAILED	The requested verification operation has failed.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_FILE	An attempt to open a file failed.
SK_ResultCode.SK_ERROR_LICENSE_SYSTEM_ IDENTIFIERS_DONT_MATCH	License system identifiers do not match.
SK_ResultCode.SK_ERROR_LICENSE_NOT_EFFECTIVE_YET	License is not yet effective.
SK_ResultCode.SK_ERROR_LICENSE_EXPIRED	License has expired.

SK_PLUS_SystemIdentifierAddCurrentIdentifier Function

Adds a SystemIdentifier element to the list of current system identifiers.

Remarks

This function only needs to be called directly when implementing custom system identifier algorithms. Otherwise, the **SK_PLUS_SystemIdentifierAlgorithmAddCurrentIdentifiers** function should be called instead.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_SystemIdentifierAddCurrentIdentifier(SK_ApiContext context, int flags,
const char *name, const char *type, const char *value);
```

Visual Basic

```
Declare Function SK_PLUS_SystemIdentifierAddCurrentIdentifier(ByVal context As Long, ByVal
flags As Long, ByVal name As String, ByVal type As String, ByVal value As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`name`

A name for the identifier being added, which must be unique or different from all other identifiers.

`type`

The type of identifier being added.

`value`

The plain-text value of the identification information.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.

SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS_SystemIdentifierAlgorithmAddCurrentIdentifiers Function

Adds SystemIdentifier elements to the list of current system identifiers using the specified, built-in algorithm.

Remarks

Important Note

In versions 5.13.3.0 and earlier of PLUSNative, an older macOS implementation of the **SK_SystemIdentifierAlgorithm.SK_SYSTEM_IDENTIFIER_ALGORITHM_HARD_DISK_VOLUME_SERIAL** algorithm was used. To retain compatibility with the older macOS implementation (which may be necessary to avoid requiring your customers to reactivate after upgrading the PLUSNative library), you can specify the **SK_FLAGS_SYSTEMIDENTIFIER_HARDDISKVOLUMESERIALS_LEGACYOSX** flag.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_SystemIdentifierAlgorithmAddCurrentIdentifiers(SK_ApiContext context,
int flags, SK_SystemIdentifierAlgorithm algorithm, const char *algorithmDataString, int
*count);
```

Visual Basic

```
Declare Function SK_PLUS_SystemIdentifierAlgorithmAddCurrentIdentifiers(ByVal context As
Long, ByVal flags As Long, ByVal algorithm As Long, ByVal algorithmDataString As String,
ByRef count As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`algorithm`

The **SK_SystemIdentifierAlgorithm** value, which identifies the algorithm that will be used when computing the system identifier data.

`algorithmDataString`

May be used in the future as a plain-text, string value to send to certain algorithms. Pass NULL or zero (0) for this parameter.

`count`

Reference/pointer to an integer, which will point to the number of system identifiers added when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_SystemIdentifierCompare Function

Compares the current system's identifiers against the identifiers authorized in the license.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_SystemIdentifierCompare(SK_ApiContext context, int flags, const char *type, int *count, int *matches);
```

Visual Basic

```
Declare Function SK_PLUS_SystemIdentifierCompare(ByVal context As Long, ByVal flags As Long, ByVal type As String, ByRef count As Long, ByRef matches As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`type`

The type of identifier to compare. Leave empty to compare all types. Other possible values include: "ComputerNameIdentifier", "HardDiskVolumeSerialIdentifier", "NicIdentifier", "NetworkNameIdentifier", or "UserNameIdentifier".

`count`

Reference/pointer to an integer, which will point to the number of system identifiers authorized in the license when the call succeeds.

`matches`

Reference/pointer to an integer, which will point to the number of identifiers authorized in the license which match the current system's identifiers when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.

SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_SystemIdentifierCurrentGetContents Function

Retrieves the contents of the current system's identifiers.

Remarks

Important Note

The XML document returned in the identifiers parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_SystemIdentifierCurrentGetContents(SK_ApiContext context, int flags,
SK_XmlDoc *identifiers);
```

Visual Basic

```
Declare Function SK_PLUS_SystemIdentifierCurrentGetContents(ByVal context As Long, ByVal
flags As Long, ByRef identifiers As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`identifiers`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle which contains the system identifiers when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.

SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS_SystemIdentifierCurrentSetContents Function

Sets the contents of the current system's identifiers.

Syntax

C/C++

```
SK_ResultCode SK_PLUS_SystemIdentifierCurrentSetContents(SK_ApiContext context, int flags,
SK_XmlDoc identifiers);
```

Visual Basic

```
Declare Function SK_PLUS_SystemIdentifierCurrentSetContents(ByVal context As Long, ByVal
flags As Long, ByVal identifiers As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`identifiers`

An XML document which contains the identifiers.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS4_GenerateUserCode1Value Function

Generates a Protection PLUS 4 compatible trigger code "User Code 1" (or Session Code) value, which may be used to make each activation attempt and each trigger code issued for a system unique.

Remarks

This function is compatible with **pp_cenum**.

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_GenerateUserCode1Value(int flags, int *userCode1);
```

Visual Basic

```
Declare Function SK_PLUS4_GenerateUserCode1Value(ByVal flags As Long, ByRef userCode1 As
Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`userCode1`

Reference/pointer to an integer, which will point to the "User Code 1" (or Session Code) value when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_DATETIME_CONVERSION_FAILED	A date-time conversion operation failed.

SK_PLUS4_GenerateUserCode2Value Function

Generates a Protection PLUS 4 compatible trigger code "User Code 2" (or "Computer ID") value from the current system identifiers.

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_GenerateUserCode2Value(SK_ApiContext context, int flags, int *user-
Code2);
```

Visual Basic

```
Declare Function SK_PLUS4_GenerateUserCode2Value(ByVal context As Long, ByVal flags As Long,
ByRef userCode2 As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`userCode2`

Reference/pointer to an integer, which will point to the "User Code 2" (or "Computer ID") value when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_GetLicenseStatusRequest Function

Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_GetLicenseStatus** and **SK_GetLicenseStatusEx** functions.

Remarks

In general, if you are using SOLO Server, you should be using **SK_SOLO_CheckInstallationStatusGetRequest** with **SK_CallXmlWebService** for status checks. The same information can also be retrieved from SOLO Server using **XmlLicenseServer's InfoCheck method**.

After calling this function to generate the URL needed, you can submit the request to the server using the **SK_HttpRequest** function, and parse the server's response using **SK_PLUS4_GetLicenseStatusResponse**

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_GetLicenseStatusRequest(SK_ApiContext context, int flags, const char *url, int licenseId, const char *password, int productId, char **requestUrl);
```

Visual Basic

```
Declare Function SK_PLUS4_GetLicenseStatusRequest(ByVal context As Long, ByVal flags As Long, ByVal url As String, ByVal licenseId As Long, ByVal password As String, ByVal productId As Long, ByRef requestUrl As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`url`

The URL to the Automation Client script. Use "secure.soft-warekey.com/solo/customers/getlicensestatus.asp" for Instant SOLO Server.

`licenseId`

The License ID used when activating the protected application.

`password`

The password used when activating the protected application.

`productId`

The Product ID for which the license was created (optional).

`requestUrl`

Reference/pointer to a string pointer, which will point to request URL that contains the query string data needed to submit the request. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_GetLicenseStatusResponse Function

Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_GetLicenseStatus** and **SK_GetLicenseStatusEx** functions.

Remarks

In general, if you are using SOLO Server, you should be using **SK_SOLO_CheckInstallationStatusGetRequest** with **SK_CallXmlWebService** for status checks. The same information can also be retrieved from SOLO Server using **XmlLicenseServer's InfoCheck method**.

Before calling this function, you need to call **SK_PLUS4_GetLicenseStatusRequest** to generate the request URL, and submit the request to the server using **SK_HttpRequest** (which returns the response this function is meant to parse).

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_GetLicenseStatusResponse(SK_ApiContext context, int flags, const char *response, int *errorCode, char **licenseStatus, int *licenseReplacedBy, char **licenseUpdate);
```

Visual Basic

```
Declare Function SK_PLUS4_GetLicenseStatusResponse(ByVal context As Long, ByVal flags As Long, ByVal response As String, ByRef errorCode As Long, ByRef licenseStatus As String, ByRef licenseReplacedBy As Long, ByRef licenseUpdate As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`response`

The response returned from the server.

`errorCode`

Reference/pointer to an integer, which will receive the Automation Client compatible error code.

`licenseStatus`

Reference/pointer to a string pointer, which will point to the string which contains the license status returned from the server when the call succeeds. The pointer must be initialized to NULL (0) prior to calling this function.

`licenseReplacedBy`

Reference/pointer to an integer, which will point to the License ID which replaced the one provided in the request if applicable.

`licenseUpdate`

Reference/pointer to a string pointer, which will point to the string which contains the license update content returned from the server when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_WEBSERVICE_RETURNED_FAILURE	A call to a Web Service succeeded, but the functionality of the Web Service returned an indicator of failure.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_GetRegDataRequest Function

Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_GetRegData** and **SK_GetRegDataEx** functions.

Remarks

In general, if you are using SOLO Server, all of the information provided by this script is included with the license files returned by SOLO Server when using **SK_SOLO_ActivateInstallationGetRequest** with **SK_CallXmlWebService** for activation. The same information can also be retrieved from SOLO Server using **XmlLicenseServer's InfoCheck method**.

After calling this function to generate the URL needed, you can submit the request to the server using the **SK_HttpRequest** function, and parse the server's response using **SK_PLUS4_GetRegDataResponse**.

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_GetRegDataRequest(SK_ApiContext context, int flags, const char *url,
int licenseId, const char *password, char **requestUrl);
```

Visual Basic

```
Declare Function SK_PLUS4_GetRegDataRequest(ByVal context As Long, ByVal flags As Long, ByVal
url As String, ByVal licenseId As Long, ByVal password As String, ByRef requestUrl As String)
As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`url`

The URL to the Automation Client script. Use "secure.softwarekey.com/solo/customers/getregdata.asp" for Instant SOLO Server.

`licenseId`

The License ID used when activating the protected application.

`password`

The password used when activating the protected application.

`requestUrl`

Reference/pointer to a string pointer, which will point to request URL that contains the query string data needed to submit the request. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_GetRegDataResponse Function

Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_GetRegData** and **SK_GetRegDataEx** functions.

Remarks

In general, if you are using SOLO Server, all of the information provided by this script is included with the license files returned by SOLO Server when using **SK_SOLO_ActivateInstallationGetRequest** with **SK_CallXmlWebService** for activation. The same information can also be retrieved from SOLO Server using **XmlLicenseServer's InfoCheck method**.

Before calling this function, you need to call **SK_PLUS4_GetRegDataRequest** to generate the request URL, and submit the request to the server using **SK_HttpRequest** (which returns the response this function is meant to parse).

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_GetRegDataResponse(SK_ApiContext context, int flags, const char *response, int *errorCode, char **company, char **contact, char **email, char **phone, char **userDefinedString1, char **userDefinedString2, char **userDefinedString3, char **userDefinedString4, char **userDefinedString5);
```

Visual Basic

```
Declare Function SK_PLUS4_GetRegDataResponse(ByVal context As Long, ByVal flags As Long, ByVal response As String, ByRef errorCode As Long, ByRef company As String, ByRef contact As String, ByRef email As String, ByRef phone As String, ByRef userDefinedString1 As String, ByRef userDefinedString2 As String, ByRef userDefinedString3 As String, ByRef userDefinedString4 As String, ByRef userDefinedString5 As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`response`

The response returned from the server.

`errorCode`

Reference/pointer to an integer, which will receive the Automation Client compatible error code.

`company`

Reference/pointer to a string pointer, which will point to the string which contains the company name returned from the server when the call succeeds.

`contact`

Reference/pointer to a string pointer, which will point to the string which contains the customer's contact name returned from the server when the call succeeds.

`email`

Reference/pointer to a string pointer, which will point to the string which contains the customer's email address returned from the server when the call succeeds.

`phone`

Reference/pointer to a string pointer, which will point to the string which contains the customer's phone number returned from the server when the call succeeds.

`userDefinedString1`

Reference/pointer to a string pointer, which will point to the string which contains the user-defined string 1 value returned from the server when the call succeeds.

`userDefinedString2`

Reference/pointer to a string pointer, which will point to the string which contains the user-defined string 2 value returned from the server when the call succeeds.

`userDefinedString3`

Reference/pointer to a string pointer, which will point to the string which contains the user-defined string 3 value returned from the server when the call succeeds.

`userDefinedString4`

Reference/pointer to a string pointer, which will point to the string which contains the user-defined string 4 value returned from the server when the call succeeds.

`userDefinedString5`

Reference/pointer to a string pointer, which will point to the string which contains the user-defined string 5 value returned from the server when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_WEBSERVICE_RETURNED_FAILURE	A call to a Web Service succeeded, but the functionality of the Web Service returned an indicator of failure.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_GetTcDataRequest Function

Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_GetTCData** and **SK_GetTCDataEx** functions.

Remarks

In general, if you are using SOLO Server, you should be using **SK_SOLO_ActivateInstallationGetRequest** with **SK_CallXmlWebService** for activation.

After calling this function to generate the URL needed, you can submit the request to the server using the **SK_HttpRequest** function, and parse the server's response using **SK_PLUS4_GetTcDataResponse**.

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_GetTcDataRequest(SK_ApiContext context, int flags, const char *url,
int licenseId, const char *password, int userCode1, int userCode2, int productId, int pro-
ductOptionId, char **requestUrl);
```

Visual Basic

```
Declare Function SK_PLUS4_GetTcDataRequest(ByVal context As Long, ByVal flags As Long, ByVal
url As String, ByVal licenseId As Long, ByVal password As String, ByVal userCode1 As Long,
ByVal userCode2 As Long, ByVal productId As Long, ByVal productOptionId As Long, ByRef
requestUrl As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`url`

The URL to the Automation Client script. Use "secure.softwarekey.com/solo/unlock/getcode.asp" for Instant SOLO Server.

`licenseId`

The License ID used when activating the protected application.

`password`

The password used when activating the protected application.

`userCode1`

The randomized User Code 1 value (also referred to as the session code). This value can be generated using the **SK_PLUS4_GenerateUserCode1Value** function.

`userCode2`

The User Code 2 value (also referred to as the Computer ID). This value can be generated using the **SK_PLUS4_GenerateUserCode2Value** function.

`productId`

The Product ID for which the license was created (optional).

`productOptionId`

The Product Option ID for which the license was created (optional).

`requestUrl`

Reference/pointer to a string pointer, which will point to request URL that contains the query string data needed to submit the request. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_GetTcDataResponse Function

Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_GetTCData** and **SK_GetTCDataEx** functions.

Remarks

In general, if you are using SOLO Server, you should be using **SK_SOLO_ActivateInstallationGetRequest** with **SK_CallXmlWebService** for activation.

Before calling this function, you need to call **SK_PLUS4_GetTcDataRequest** to generate the request URL, and submit the request to the server using **SK_HttpRequest** (which returns the response this function is meant to parse).

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_GetTcDataResponse(SK_ApiContext context, int flags, const char
*response, int *errorCode, int *activationCode1, int *activationCode2, char **licenseUpdate);
```

Visual Basic

```
Declare Function SK_PLUS4_GetTcDataResponse(ByVal context As Long, ByVal flags As Long, ByVal
response As String, ByRef errorCode As Long, ByRef activationCode1 As Long, ByRef activ-
ationCode2 As Long, ByRef licenseUpdate As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`response`

The response returned from the server.

`errorCode`

Reference/pointer to an integer, which will receive the Automation Client compatible error code.

`activationCode1`

Reference/pointer to an integer, which will point to the Activation Code 1 value returned from the server.

`activationCode2`

Reference/pointer to an integer, which will point to the Activation Code 2 value returned from the server.

`licenseUpdate`

Reference/pointer to a string pointer, which will point to the string which contains the license update content returned from the server when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_WEBSERVICE_RETURNED_FAILURE	A call to a Web Service succeeded, but the functionality of the Web Service returned an indicator of failure.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_NDecrypt Function

Decrypts a number or RegKey2 value.

Remarks

This function is compatible with **pp_ndecrypt**.

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_NDecrypt(int flags, int number, int seed, int *result);
```

Visual Basic

```
Declare Function SK_PLUS4_NDecrypt(ByVal flags As Long, ByVal number As Long, ByVal seed As Long, ByRef result As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`number`

The number to decrypt.

`seed`

The encryption seed, which must be a value between 1 and 255. Larger values are truncated to the first byte.

`result`

Reference/pointer to an integer, which will receive the decrypted value when the call succeeds or '-1' if the call fails.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS4_NEncrypt Function

Encrypts a number or RegKey2 value.

Remarks

This function is compatible with **pp_nencrypt**.

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_NEncrypt(int flags, int number, int seed, int *result);
```

Visual Basic

```
Declare Function SK_PLUS4_NEncrypt(ByVal flags As Long, ByVal number As Long, ByVal seed As
Long, ByRef result As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`number`

The number to encrypt, which may only be a value between 0 and 16383.

`seed`

The encryption seed, which must be a value between 1 and 255. Larger values are truncated to the first byte.

`result`

Reference/pointer to an integer, which will receive the encrypted value when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_PLUS4_PostEvalDataRequest Function

Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_PostEvalData** and **SK_PostEvalDataEx** functions.

Remarks

After calling this function to generate the URL needed, you can submit the request to the server using the **SK_HttpRequest** function (you can use a URL of "secure.softwarekey.com/solo/products/trialsignup.asp" for Instant SOLO Server), and parse the server's response using **SK_PLUS4_PostEvalDataResponse**.

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_PostEvalDataRequest(SK_ApiContext context, int flags, int productId,
const char *email, const char *companyName, int distributorId, const char *firstName, const
char *lastName, const char *address1, const char *address2, const char *city, const char
*stateProvince, const char *postalCode, const char *country, const char *phone, const char
*source, char **postString);
```

Visual Basic

```
Declare Function SK_PLUS4_PostEvalDataRequest(ByVal context As Long, ByVal flags As Long,
ByVal productId As Long, ByVal email As String, ByVal companyName As String, ByVal dis-
tributorId As Long, ByVal firstName As String, ByVal lastName As String, ByVal address1 As
String, ByVal address2 As String, ByVal city As String, ByVal stateProvince As String, ByVal
postalCode As String, ByVal country As String, ByVal phone As String, ByVal source As String,
ByRef postString As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`productId`

The Product ID of the protected application which is being evaluated.

`email`

The customer's email address.

`companyName`

The customer's company name (optional).

`distributorId`

The Distributor ID of the company which originated the evaluation/lead.

`firstName`

The customer's first/given name (optional).

`lastName`

The customer's last/family name (optional).

`address1`

The first line of the customer's street address (optional).

`address2`

The second line of the customer's street address (optional).

`city`

The city in which the customer is located (optional).

`stateProvince`

The state/province in which the customer is located (optional).

`postalCode`

The customer's zip/postal code (optional).

`country`

The country in which the customer is located (optional).

`phone`

The customer's phone number (optional).

`source`

The source of the lead, or how the customer heard about the protected application (e.g. Internet search, friend, etc...).

`postString`

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_PostEvalDataResponse Function

Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_PostRegData** or **SK_PostRegDataEx** functions.

Remarks

Before calling this function, you need to call **SK_PLUS4_PostEvalDataRequest** to generate the URL-encoded post string, and submit the request to the server using **SK_HttpRequest** (which returns the response this function is meant to parse).

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_PostEvalDataResponse(SK_ApiContext context, int flags, const char *response, int *errorCode, int *registrationId);
```

Visual Basic

```
Declare Function SK_PLUS4_PostEvalDataResponse(ByVal context As Long, ByVal flags As Long, ByVal response As String, ByRef errorCode As Long, ByRef registrationId As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`response`

The response returned from the server.

`errorCode`

Reference/pointer to an integer, which will receive the Automation Client compatible error code.

`registrationId`

Reference/pointer to an integer, which will receive the unique identifier issued by the server after successfully registering an evaluation.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_WEBSERVICE_RETURNED_FAILURE	A call to a Web Service succeeded, but the functionality of the Web Service returned an indicator of failure.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_PostRegDataRequest Function

Generates a URL with the query string parameters needed for calling server-side scripts which were originally designed to work with the Automation Client's **SK_PostRegData** > function.

Remarks

The same information can also be updated in SOLO Server using **XmlLicenseServer's UpdateRegistration method**.

After calling this function to generate the URL needed, you can submit the request to the server using the **SK_HttpRequest** function (use a URL of "secure.softwarekey.com/solo/postings/postregdata.asp" for Instant SOLO Server), and parse the server's response using **SK_PLUS4_PostRegDataResponse**.

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_PostRegDataRequest(SK_ApiContext context, int flags, int licenseId,
const char *password, const char *companyName, const char *firstName, const char *lastName,
const char *address1, const char *address2, const char *city, const char *stateProvince,
const char *postalCode, const char *country, const char *phone, const char *fax, const char
*email, SK_BOOL overwrite, SK_BOOL notifyProduct, SK_BOOL notifyPartners, char **postString);
```

Visual Basic

```
Declare Function SK_PLUS4_PostRegDataRequest(ByVal context As Long, ByVal flags As Long,
ByVal licenseId As Long, ByVal password As String, ByVal companyName As String, ByVal
firstName As String, ByVal lastName As String, ByVal address1 As String, ByVal address2 As
String, ByVal city As String, ByVal stateProvince As String, ByVal postalCode As String,
ByVal country As String, ByVal phone As String, ByVal fax As String, ByVal email As String,
ByVal overwrite As Long, ByVal notifyProduct As Long, ByVal notifyPartners As Long, ByRef
postString As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`licenseId`

The License ID used when activating the protected application.

`password`

`companyName`

The customer's company name.

`firstName`

The customer's first/given name.

`lastName`

The customer's last/family name.

`address1`

The first line of the customer's street address.

`address2`

The second line of the customer's street address.

`city`

The city in which the customer is located.

`stateProvince`

The state/province in which the customer is located.

`postalCode`

The customer's zip/postal code.

`country`

The country in which the customer is located.

`phone`

The customer's phone number.

`fax`

`email`

The customer's email address.

`overwrite`

Set to 1 (TRUE) to overwrite previous registration data, or 0 (FALSE) to prevent the call from overwriting existing registration data.

`notifyProduct`

If the customer wants to be notified of product-related events, releases, and promotions, set this to 1 (TRUE); otherwise, set this to 0 (FALSE).

`notifyPartners`

If the customer wants to be notified of partner promotions, set this to 1 (TRUE); otherwise, set this to 0 (FALSE).

`postString`

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_PostRegDataResponse Function

Parses a response from a server-side script which was originally designed to work with the Automation Client's **SK_PostRegData** function.

Remarks

The same information can also be updated in SOLO Server using **XmlLicenseServer's UpdateRegistration method**.

Before calling this function, you need to call **SK_PLUS4_PostRegDataRequest** to generate the request URL, and submit the request to the server using **SK_HttpRequest** (which returns the response this function is meant to parse).

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_PostRegDataResponse(SK_ApiContext context, int flags, const char
*response, int *errorCode);
```

Visual Basic

```
Declare Function SK_PLUS4_PostRegDataResponse(ByVal context As Long, ByVal flags As Long,
ByVal response As String, ByRef errorCode As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`response`

The response returned from the server.

`errorCode`

Reference/pointer to an integer, which will receive the Automation Client compatible error code.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_WEBSERVICE_RETURNED_FAILURE	A call to a Web Service succeeded, but the functionality of the Web Service returned an indicator of failure.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_PLUS4_ValidateTriggerCode Function

Determines whether or not a trigger code is valid.

Syntax

C/C++

```
SK_ResultCode SK_PLUS4_ValidateTriggerCode(int flags, int activationCode1, int activ-
ationCode2, int userCode1, int userCode2, int tcSeed, int dataSeed, int *code, int *tcData);
```

Visual Basic

```
Declare Function SK_PLUS4_ValidateTriggerCode(ByVal flags As Long, ByVal activationCode1 As
Long, ByVal activationCode2 As Long, ByVal userCode1 As Long, ByVal userCode2 As Long, ByVal
tcSeed As Long, ByVal dataSeed As Long, ByRef code As Long, ByRef tcData As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`activationCode1`

Activation Code 1 (RegKey 1).

`activationCode2`

Activation Code 2 (RegKey 2). Pass zero (0) to omit.

`userCode1`

User Code 1 (or Session Code).

`userCode2`

User Code 2 (or Computer ID).

`tcSeed`

Trigger Code Seed.

`dataSeed`

Event Data Seed (RegKey2 Seed, which must be a value between 1 and 255).

`code`

Reference/pointer to an integer, which receive the trigger code number (or a value between 1 and 50) when the call succeeds.

`tcData`

Reference/pointer to an integer, which will receive the decrypted trigger code event data value (or a value between 0 and 16383) when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_TRIGGER_CODE_INVALID	An invalid "Activation Code 1" value was entered by the user.
SK_ResultCode.SK_ERROR_TRIGGER_CODE_EVENT_DATA_INVALID	An invalid "Activation Code 2" value was entered by the user.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_RegistryValueGet Function

Gets a Windows Registry key's string value.

Remarks

Important Note

The string returned in the value parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_RegistryValueGet(int flags, const char *path, char **value);
```

Visual Basic

```
Declare Function SK_RegistryValueGet(ByVal flags As Long, ByVal path As String, ByRef value
As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`path`

The full path to the Windows Registry key.

`value`

Reference/pointer to the string pointer where the Windows Registry key's value will be stored. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_REGISTRY_KEY	An attempt to open a registry key failed.
SK_ResultCode.SK_ERROR_COULD_NOT_READ_REGISTRY_KEY	An attempt to read a registry key value failed.
SK_ResultCode.SK_ERROR_UNSUPPORTED_OS	The current operating system is not supported by this feature or function.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_RegistryValueSet Function

Sets a Windows Registry key's string value.

Syntax

C/C++

```
SK_ResultCode SK_RegistryValueSet(int flags, const char *path, const char *value);
```

Visual Basic

```
Declare Function SK_RegistryValueSet(ByVal flags As Long, ByVal path As String, ByVal value
As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`path`

The full path to the Windows Registry key.

`value`

The new value to store in the Windows Registry key.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_REGISTRY_KEY	An attempt to open a registry key failed.
SK_ResultCode.SK_ERROR_COULD_NOT_WRITE_REGISTRY_KEY	An attempt to write a registry key value failed.

SK_ResultCode.SK_ERROR_UNSUPPORTED_OS	The current operating system is not supported by this feature or function.

SK_SessionCodeGenerate Function

Generates a randomized "session code" (similar to a cryptographic nonce).

Remarks

Important Note

The string returned in the description parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SessionCodeGenerate(int flags, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SessionCodeGenerate(ByVal flags As Long, ByRef sessionCode As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the session code value when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_STRING_CONVERSION_FAILED	A string conversion operation failed.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_FILE	An attempt to open a file failed.

SK_ResultCode.SK_ERROR_COULD_NOT_READ_FILE	An attempt to read a file failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.
SK_ResultCode.SK_ERROR_LIBRARY_UNAVAILABLE	Required system library is missing for failed to load.
SK_ResultCode.SK_ERROR_LIBRARY_FUNCTION_UNAVAILABLE	Required library function is missing.

SK_SOLO_ActivateInstallationGetRequest Function

Builds a request to send to the XmlActivationService web service's ActivateInstallationLicenseFile method in SOLO Server.

Remarks

Important Note

The XML document returned in the request parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

The string returned in the sessionCode parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_ActivateInstallationGetRequest(SK_ApiContext context, int flags, int licenseId, const char *pw, const char *serialNum, int userCode1, int userCode2, SK_BOOL requireRegistration, const char *installationName, const char *prevInstallationId, SK_XmlDoc *request, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SOLO_ActivateInstallationGetRequest(ByVal context As Long, ByVal flags As Long, ByVal licenseId As Long, ByVal pw As String, ByVal serialNum As String, ByVal userCode1 As Long, ByVal userCode2 As Long, ByVal requireRegistration As Long, ByVal installationName As String, ByVal prevInstallationId As String, ByRef request As Long, ByRef sessionCode As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`licenseId`

The License ID issued by SOLO Server.

`pw`

The activation password for the license, or customer password, in SOLO Server.

`serialNum`

The Serial Number issued for the license in SOLO Server (leave this field empty when using the License ID and password).

`userCode1`

This is only used for issuing Protection PLUS 4 compatible Trigger Codes, and is the User-Code 1 value (or session code). Use a value of 1000 if you are not using Protection PLUS 4 compatibility features.

`userCode2`

This is only used for issuing Protection PLUS 4 compatible Trigger Codes, and is the User-Code 2 value (or Computer ID). Use a value of 1000 if you are not using Protection PLUS 4 compatibility features.

`requireRegistration`

Set to TRUE (1) when the customer must have their contact information (such as name, email, etc...) registered with SOLO Server before activation is permitted.

`installationName`

The optional, human-readable name of this installation. This may be a entered by your users, or may be a value of your choosing.

`prevInstallationId`

The optional, previous Installation ID which was last used on the system being activated.

`request`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle that contains the request when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the unique session code generated for this request when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_SOLO_CheckInstallationStatusGetRequest Function

Builds a request to send to the XmlActivationService web service's CheckInstallationStatus method in SOLO Server.

Remarks

Important Note

The XML document returned in the request parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

The string returned in the sessionCode parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_CheckInstallationStatusGetRequest(SK_ApiContext context, int flags,
const char *installationId, SK_XmlDoc *request, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SOLO_CheckInstallationStatusGetRequest(ByVal context As Long, ByVal flags
As Long, ByVal installationId As String, ByRef request As Long, ByRef sessionCode As String)
As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`installationId`

The Installation ID of the activated license.

`request`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle that contains the request when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the unique session code generated for this request when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_SOLO_DeactivateInstallationGetRequest Function

Builds a request to send to the XmlActivationService web service's DeactivateInstallation method in SOLO Server.

Remarks

Important Note

The XML document returned in the request parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

The string returned in the sessionCode parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_DeactivateInstallationGetRequest(SK_ApiContext context, int flags,
const char *installationId, SK_XmlDoc *request, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SOLO_DeactivateInstallationGetRequest(ByVal context As Long, ByVal flags
As Long, ByVal installationId As String, ByRef request As Long, ByRef sessionCode As String)
As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`installationId`

The Installation ID of the activated license.

`request`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle that contains the request when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the unique session code generated for this request when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_SOLO_GetLicenseFileGetRequest Function

Builds a request to send to the XmlLicenseFileService web service's GetLicenseFile method in SOLO Server.

Remarks

Important Note

The XML document returned in the request parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

The string returned in the sessionCode parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_GetLicenseFileGetRequest(SK_ApiContext context, int flags, const char *installationId, SK_XmlDoc *request, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SOLO_GetLicenseFileGetRequest(ByVal context As Long, ByVal flags As Long, ByVal installationId As String, ByRef request As Long, ByRef sessionCode As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`installationId`

The Installation ID of the activated license.

`request`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle that contains the request when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the unique session code generated for this request when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_SOLO_ManualRequestFileLoad Function

Loads and decrypts a manual response file.

Remarks

Important Note

The XML document returned in the response parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_ManualRequestFileLoad(SK_ApiContext context, int flags, const char *path, SK_XmlDoc *response);
```

Visual Basic

```
Declare Function SK_SOLO_ManualRequestFileLoad(ByVal context As Long, ByVal flags As Long, ByVal path As String, ByRef response As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`path`

The absolute path to the location of the manual response file.

`response`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle which contains the web service response when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_SOLO_ManualRequestFileSave Function

Saves a manual request file to the specified path.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_ManualRequestFileSave(SK_ApiContext context, int flags, const char
*url, const char *customMarkupTop, const char *customMarkupBottom, const char *path, SK_
XmlDoc request);
```

Visual Basic

```
Declare Function SK_SOLO_ManualRequestFileSave(ByVal context As Long, ByVal flags As Long,
ByVal url As String, ByVal customMarkupTop As String, ByVal customMarkupBottom As String,
ByVal path As String, ByVal request As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`url`

The SOLO Server manual request URL to post to (prefixed with http:// or https://). For Instant SOLO Server, the **SK_CONST_WEBSERVICE_MANUALREQUEST_URL** constant is provided.

`customMarkupTop`

(Optional - set to an empty string or NULL (0) to ignore.) The markup prior to the form. You may set this to implement a custom design.

`customMarkupBottom`

(Optional - set to an empty string or NULL (0) to ignore.) The markup after the form. You may set this to implement a custom design.

`path`

The absolute path to the location where the manual request file should be saved, such as the Desktop folder.

`request`

An XML document containing the contents of the web service request.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_SOLO_NetworkSessionCheckinGetRequest Function

Important

This function is preliminary, and is subject to change.

Builds a request to send to the XmlNetworkFloatingService web service's CheckinSession method in SOLO Server.

Remarks

Important Note

The XML document returned in the request parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_NetworkSessionCheckinGetRequest(SK_ApiContext context, int flags, const char *sessionId, SK_XmlDoc *request, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SOLO_NetworkSessionCheckinGetRequest(ByVal context As Long, ByVal flags As Long, ByVal sessionId As String, ByRef request As Long, ByRef sessionCode As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`sessionId`

The Session ID issued by SOLO Server.

`request`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle that contains the request when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the unique session code generated for this request when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_SOLO_NetworkSessionCheckoutGetRequest Function

Important

This function is preliminary, and is subject to change.

Builds a request to send to the XmlNetworkFloatingService web service's CheckoutSession method in SOLO Server.

Remarks

Important Note

The XML document returned in the request parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_NetworkSessionCheckoutGetRequest(SK_ApiContext context, int flags,
const char *sessionId, const char *certificatePath, double requestedCheckoutDuration, SK_
XmlDoc *request, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SOLO_NetworkSessionCheckoutGetRequest(ByVal context As Long, ByVal flags
As Long, ByVal sessionId As String, ByVal certificatePath As String, ByVal reques-
tedCheckoutDuration As Long, ByRef request As Long, ByRef sessionCode As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`sessionId`

The Session ID issued by SOLO Server.

`certificatePath`

The absolute path to the Network Session Certificate file.

`requestedCheckoutDuration`

The amount of time (in hours) we would like the requested checkout to last.

`request`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle that contains the request when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the unique session code generated for this request when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_SOLO_NetworkSessionCloseGetRequest Function

Important

This function is preliminary, and is subject to change.

Builds a request to send to the XmlNetworkFloatingService web service's CloseSession method in SOLO Server.

Remarks

Important Note

The XML document returned in the request parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_NetworkSessionCloseGetRequest(SK_ApiContext context, int flags, const char *sessionId, SK_XmlDoc *request, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SOLO_NetworkSessionCloseGetRequest(ByVal context As Long, ByVal flags As Long, ByVal sessionId As String, ByRef request As Long, ByRef sessionCode As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`sessionId`

The Session ID issued by SOLO Server.

`request`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle that contains the request when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the unique session code generated for this request when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_SOLO_NetworkSessionOpenGetRequest Function

Important

This function is preliminary, and is subject to change.

Builds a request to send to the XmlNetworkFloatingService web service's OpenSession method in SOLO Server.

Remarks

Important Note

The XML document returned in the request parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_NetworkSessionOpenGetRequest(SK_ApiContext context, int flags, int licenseId, const char *pw, SK_XmlDoc *request, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SOLO_NetworkSessionOpenGetRequest(ByVal context As Long, ByVal flags As Long, ByVal licenseId As Long, ByVal pw As String, ByRef request As Long, ByRef sessionCode As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`licenseId`

The License ID issued by SOLO Server.

`pw`

The activation password for the license, or customer password, in SOLO Server.

`request`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle that contains the request when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the unique session code generated for this request when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_SOLO_NetworkSessionPollGetRequest Function

Important

This function is preliminary, and is subject to change.

Builds a request to send to the XmlNetworkFloatingService web service's PollSession method in SOLO Server.

Remarks

Important Note

The XML document returned in the request parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_SOLO_NetworkSessionPollGetRequest(SK_ApiContext context, int flags, const char *sessionId, const char* certificatePath, SK_XmlDoc *request, char **sessionCode);
```

Visual Basic

```
Declare Function SK_SOLO_NetworkSessionPollGetRequest(ByVal context As Long, ByVal flags As Long, ByVal sessionId As String, ByVal certificatePath As String, ByRef request As Long, ByRef sessionCode As String) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`sessionId`

The Session ID issued by SOLO Server.

`certificatePath`

The absolute path to the Network Session Certificate file.

`request`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle that contains the request when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

`sessionCode`

Reference/pointer to a string pointer, which will point to the string which contains the unique session code generated for this request when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_StringConvertMultiByteStringToWideString Function

Converts a string with the specified encoding to a wide string.

Remarks

Important Note

The wide string returned in the result parameter must be freed from memory! The **SK_StringDisposeW** function is recommended for this purpose.

Important Note

When using wide-strings, be careful to ensure that the correct encoding is used. UTF-16 (2-byte) characters should be used on Windows, and UTF-32 (4-byte) characters should be used on Linux and OS X.

Syntax

C/C++

```
SK_ResultCode SK_StringConvertMultiByteStringToWideString(int flags, SK_StringEncoding encoding, const char *source, wchar_t **result);
```

Visual Basic

```
'Unsupported.
```

Arguments

`flags`

Any **flags** passed into this function-call.

`encoding`

The **SK_StringEncoding** value, which identifies the encoding of the source string.

`source`

The source string to convert.

`result`

Reference/pointer to a wide string pointer, which will point to the wide string which contains the converted string when the call succeeds. The wide string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_STRING_CONVERSION_FAILED	A string conversion operation failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_StringConvertWideStringToMultiByteString Function

Converts a wide string to a string using the specified encoding.

Remarks

Important Note

The string returned in the result parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Important Note

When using wide-strings, be careful to ensure that the correct encoding is used. UTF-16 (2-byte) characters should be used on Windows, and UTF-32 (4-byte) characters should be used on Linux and OS X.

Syntax

C/C++

```
SK_ResultCode SK_StringConvertWideStringToMultiByteString(int flags, SK_StringEncoding encod-
ing, const wchar_t *source, char **result);
```

Visual Basic

```
'Unsupported.
```

Arguments

`flags`

Any **flags** passed into this function-call.

`encoding`

The **SK_StringEncoding** value, which identifies the desired encoding of the result string.

`source`

The source string to convert.

`result`

Reference/pointer to a string pointer, which will point to the string which contains the converted string when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_STRING_CONVERSION_FAILED	A string conversion operation failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_StringCopyToBuffer Function

Copies a string to a fixed-size string buffer.

Syntax

C/C++

```
SK_ResultCode SK_StringCopyToBuffer(int flags, const char *source, char *destination, int destinationSize);
```

Visual Basic

```
Declare Function SK_StringCopyToBuffer(ByVal flags As Long, ByVal source As String, ByVal destination As String, ByVal destinationSize As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`source`

The source string, which will be copied to the fixed-size destination buffer.

`destination`

The destination string buffer which will contain the copy of the source string.

`destinationSize`

The size of the destination string buffer.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_StringCopyToBufferW Function

Copies a wide string to a fixed-size wide string buffer.

Remarks

Important Note

When using wide-strings, be careful to ensure that the correct encoding is used. UTF-16 (2-byte) characters should be used on Windows, and UTF-32 (4-byte) characters should be used on Linux and OS X.

Syntax

C/C++

```
SK_ResultCode SK_StringCopyToBufferW(int flags, const wchar_t *source, wchar_t *destination,
int destinationSize);
```

Visual Basic

```
'Unsupported. Use SK_StringCopyToBuffer instead.
```

Arguments

`flags`

Any **flags** passed into this function-call.

`source`

The source string, which will be copied to the fixed-size destination buffer.

`destination`

The destination string buffer which will contain the copy of the source string.

`destinationSize`

The size, in characters, of the destination string buffer.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_StringDispose Function

Disposes a string, which clears it from memory and sets it's pointer to NULL (0).

Syntax

C/C++

```
SK_ResultCode SK_StringDispose(int flags, char **ptr);
```

Visual Basic

```
Declare Function SK_StringDispose(ByVal flags As Long, ByRef ptr As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`ptr`

Reference/pointer to a string pointer.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_StringDisposeW Function

Disposes a wide string, which clears it from memory and sets it's pointer to NULL (0).

Remarks

Important Note

When using wide-strings, be careful to ensure that the correct encoding is used. UTF-16 (2-byte) characters should be used on Windows, and UTF-32 (4-byte) characters should be used on Linux and OS X.

Syntax

C/C++

```
SK_ResultCode SK_StringDisposeW(int flags, wchar_t **ptr);
```

Visual Basic

```
'Unsupported. Use SK_StringDispose instead.
```

Arguments

`flags`

Any **flags** passed into this function-call.

`ptr`

Reference/pointer to a wide string pointer.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_StringGetLength Function

Gets the length of the specified string.

Syntax

C/C++

```
int SK_StringGetLength(const char *source);
```

Visual Basic

```
Declare Function SK_StringGetLength(ByVal source As String) As Long
```

Arguments

`source`

A pointer to the string to get the length of.

Returns

The length of the string, in characters, not including the NULL terminus.

SK_StringGetLengthW Function

Gets the length of the specified wide string.

Remarks

Important Note

When using wide-strings, be careful to ensure that the correct encoding is used. UTF-16 (2-byte) characters should be used on Windows, and UTF-32 (4-byte) characters should be used on Linux and OS X.

Syntax

C/C++

```
int SK_StringGetLengthW(const wchar_t *source);
```

Visual Basic

```
'Unsupported. Use SK_StringGetLength instead.
```

Arguments

`source`

A pointer to the wide string to get the length of.

Returns

The length of the wide string, in characters, not including the NULL terminus.

SK_StringToVersionNumbers Function

Parses the individual version number parts from a version string formatted like X.X.X.X, where each X is a positive integer value no longer than 5 digits in length.

Remarks

Passing a NULL to any of the version part parameters (major, minor, revision, build) will cause the parameter to be ignored.

Syntax

C/C++

```
SK_ResultCode SK_StringToVersionNumbers(int flags, const char *version, int *major, int *minor, int *revision, int *build);
```

Visual Basic

```
Declare Function SK_StringToVersionNumbers(ByVal flags As Long, ByVal version As String, ByRef major As Long, ByRef minor As Long, ByRef revision As Long, ByRef build As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`version`

The version string to parse.

`major`

The major (first) version number part parsed from the string.

`minor`

The minor (second) version number part parsed from the string.

`revision`

The revision (third) version number part parsed from the string.

`build`

The build (fourth) version number part parsed from the string.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_SystemRemoteSessionDetect Function

Detects whether or not the protected application is being run in a remote session via Terminal Services/Remote Desktop.

Remarks

This function is presently only supported in Windows desktop environments. Calling this function on other, unsupported platforms will result in **SK_ResultCode.SK_ERROR_UNSUPPORTED_OS** being returned. The only type of remote session this function detects is a Terminal Services/Remote Desktop session. This function also serves as an equivalent to calling **pp_sysinfo(PP_TERMSERV)** in Protection PLUS 4.

Syntax

C/C++

```
SK_ResultCode SK_SystemRemoteSessionDetect(SK_ApiContext context, int flags, SK_RemoteSessionTypes *sessionDetected);
```

Visual Basic

```
Declare Function SK_SystemRemoteSessionDetect(ByVal context As Long, ByVal flags As Long, ByRef sessionDetected As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`sessionDetected`

Reference/pointer to an integer, which will point to a **SK_RemoteSessionTypes** value indicating what (if any) remote session was detected.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ResultCode.SK_ERROR_LIBRARY_FUNCTION_UNAVAILABLE	Required library function is missing.

SK_SystemVirtualMachineDetect Function

Detects whether or not the protected application is being run in a virtual machine guest environment.

Remarks

Reference the **SK_VirtualMachineTypes** enumeration for a list of hypervisors this function is designed to detect.

Failure to detect a virtual machine guest environment does not guarantee that the application is not running in one. This function simply makes an effort to detect the most common hypervisors while avoiding the possibility of running into false-positives. The guest environments this can detect include:

- Linux guests: HyperV, Parallels, VirtualBox, VMware, XenServer.
- macOS guests: VirtualBox, VMware.
- Windows guests: HyperV, Parallels, VirtualBox, VirtualPC, VMware, XenServer.

If calling this function causes a long or indefinite hang in your application, you may need to pass the **SK_FLAGS_DISABLE_WMI** flag to prevent this function from using WMI queries. While this kind of issue is not very common, it is especially possible to encounter when the protected application is a service that starts automatically, and does not have the Windows Management Instrumentation (WMI) service configured as a dependency.

Syntax

C/C++

```
SK_ResultCode SK_SystemVirtualMachineDetect(SK_ApiContext context, int flags, SK_VirtualMachineTypes *vmDetected);
```

Visual Basic

```
Declare Function SK_SystemVirtualMachineDetect(ByVal context As Long, ByVal flags As Long, ByRef vmDetected As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`vmDetected`

Reference/pointer to an integer, which will point to a **SK_VirtualMachineTypes** value indicating what (if any) virtual machine guest environment was detected.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_PLATFORM_ERROR	Platform specific API or system call fails.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.
SK_ResultCode.SK_ERROR_LIBRARY_FUNCTION_UNAVAILABLE	Required library function is missing.

SK_XmlDocumentCreateFromString Function

Creates an XML document from a string representation.

Remarks

Important Note

The XML document returned in the xml parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_XmlDocumentCreateFromString(int flags, const char *input, SK_XmlDoc *xml);
```

Visual Basic

```
Declare Function SK_XmlDocumentCreateFromString(ByVal flags As Long, ByVal input As String,
ByRef xml As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`input`

A string containing XML content from which the document will be created.

`xml`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_XmlDocumentDecryptRsa Function

Decrypts an XML document using the RSA Algorithm.

Remarks

Important Note

The XML document returned in the plainText parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_XmlDocumentDecryptRsa(SK_ApiContext context, int flags, SK_BOOL selfSigned,
SK_XmlDoc cipherText, SK_XmlDoc *plainText);
```

Visual Basic

```
Declare Function SK_XmlDocumentDecryptRsa(ByVal context As Long, ByVal flags As Long, ByVal
selfSigned As Long, ByVal cipherText As Long, ByRef plainText As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`selfSigned`

When set to TRUE (1), the Client Key will be used to decrypt and verify the document.

`cipherText`

The encrypted document containing the cipher text to decrypt.

`plainText`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle which contains the decrypted plain-text contents when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_DECRYPTION_FAILED	The requested decryption operation has failed.
SK_ResultCode.SK_ERROR_VERIFICATION_FAILED	The requested verification operation has failed.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_XmlDocumentDispose Function

Disposes an XML document, which clears it from memory and sets it's pointer to NULL (0).

Syntax

C/C++

```
SK_ResultCode SK_XmlDocumentDispose(int flags, SK_XmlDoc *xml);
```

Visual Basic

```
Declare Function SK_XmlDocumentDispose(ByVal flags As Long, ByRef xml As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

Reference/pointer to a **SK_XmlDoc** handle, which points to the XML document to dispose.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlDocumentEncryptRsa Function

Encrypts an XML document using the RSA Algorithm.

Remarks

Important Note

The XML document returned in the cipherText parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_XmlDocumentEncryptRsa(SK_ApiContext context, int flags, SK_BOOL selfSign,
SK_XmlDoc plainText, SK_XmlDoc *cipherText);
```

Visual Basic

```
Declare Function SK_XmlDocumentEncryptRsa(ByVal context As Long, ByVal flags As Long, ByVal
selfSign As Long, ByVal plainText As Long, ByRef cipherText As Long) As Long
```

Arguments

`context`

The **API Context** handle.

`flags`

Any **flags** passed into this function-call.

`selfSign`

When set to TRUE (1), the Client Key will be used to encrypted and sign the document.

`plainText`

The plain-text document to encrypt.

`cipherText`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle which contains the encrypted contents when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_ENCRYPTION_FAILED	The requested encryption operation has failed.
SK_ResultCode.SK_ERROR_SIGNING_FAILED	The requested signing operation has failed.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_XmlDocumentGetDocumentString Function

Retrieves the raw contents of the XML document.

Remarks

Important Note

The string returned in the content parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_XmlDocumentGetDocumentString(int flags, SK_XmlDoc xml, char **content);
```

Visual Basic

```
Declare Function SK_XmlDocumentGetDocumentString(ByVal flags As Long, ByVal xml As Long,
ByRef content As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`content`

Reference/pointer to a string pointer, which will point to the string which contains the XML document's contents when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.

SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_XmlDocumentLoadFile Function

Loads an XML document from a file or registry value.

Remarks

Important Note

The XML document returned in the xml parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_XmlDocumentLoadFile(int flags, const char *path, SK_XmlDoc *xml);
```

Visual Basic

```
Declare Function SK_XmlDocumentLoadFile(ByVal flags As Long, ByVal path As String, ByRef xml
As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`path`

The absolute path to a file or registry value.

`xml`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_FILE	An attempt to open a file failed.
SK_ResultCode.SK_ERROR_COULD_NOT_READ_FILE	An attempt to read a file failed.
SK_ResultCode.SK_ERROR_COULD_NOT_OPEN_REGISTRY_KEY	An attempt to open a registry key failed.
SK_ResultCode.SK_ERROR_COULD_NOT_READ_REGISTRY_KEY	An attempt to read a registry key value failed.
SK_ResultCode.SK_ERROR_IO_OPERATION_FAILED	An attempt to perform an I/O operation failed.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_XmlElementAddNew Function

Adds an element to a given XML document.

Syntax

C/C++

```
SK_ResultCode SK_XmlElementAddNew(int flags, SK_XmlDoc xml, const char *xpath, const char *element, const char *value);
```

Visual Basic

```
Declare Function SK_XmlElementAddNew(ByVal flags As Long, ByVal xml As Long, ByVal xpath As String, ByVal element As String, ByVal value As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document to which the child element will be appended.

`element`

The new element's tag name.

`value`

The new element's value.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlNodeGetDocument Function

Creates an new XML document from a sub-document in the specified XML document.

Remarks

Important Note

The XML document returned in the output parameter must be freed from memory! The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeGetDocument(int flags, SK_XmlDoc input, const char *xpath, SK_XmlDoc *output);
```

Visual Basic

```
Declare Function SK_XmlNodeGetDocument(ByVal flags As Long, ByVal input As Long, ByVal xpath As String, ByRef output As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`input`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`output`

Reference/pointer to a **SK_XmlDoc** handle, which will point to the XML document handle when the call succeeds. The handle must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlNodeGetValueDateTimeString Function

Retrieves a date-time string value from a given XML node.

Remarks

Important Note

The string returned in the value parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeGetValueDateTimeString(int flags, SK_XmlDoc xml, const char *xpath,
char **value);
```

Visual Basic

```
Declare Function SK_XmlNodeGetValueDateTimeString(ByVal flags As Long, ByVal xml As Long,
ByVal xpath As String, ByRef value As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`value`

Reference/pointer to a string pointer, which will point to the string which contains the node's text value when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlNodeGetValueDouble Function

Retrieves a double floating point value from a given XML node.

Remarks

Important Note

When using this function in an application which uses vfork on POSIX-compatible platforms other than Windows and OS X, you should consider using the **SK_FLAGS_STRING_CULTURE_USE_CURRENT** flag to avoid unexpected behavior.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeGetValueDouble(int flags, SK_XmlDoc xml, const char *xpath, double *value);
```

Visual Basic

```
Declare Function SK_XmlNodeGetValueDouble(ByVal flags As Long, ByVal xml As Long, ByVal xpath As String, ByRef value As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`value`

Reference/pointer to a double, which will point to the node's value when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_XML_NODE_MISSING	The requested XML node could not be found.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlNodeGetValueInt Function

Retrieves an integer value from a given XML node.

Remarks

Important Note

When using this function in an application which uses vfork on POSIX-compatible platforms other than Windows and OS X, you should consider using the **SK_FLAGS_STRING_CULTURE_USE_CURRENT** flag to avoid unexpected behavior.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeGetValueInt(int flags, SK_XmlDoc xml, const char *xpath, int *value);
```

Visual Basic

```
Declare Function SK_XmlNodeGetValueInt(ByVal flags As Long, ByVal xml As Long, ByVal xpath As String, ByRef value As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`value`

Reference/pointer to an integer, which will point to the node's value when the call succeeds.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlNodeGetValueString Function

Retrieves a string value from a given XML node.

Remarks

Important Note

The string returned in the value parameter must be freed from memory! The **SK_StringDispose** function is recommended for this purpose.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeGetValueString(int flags, SK_XmlDoc xml, const char *xpath, SK_BOOL includeInnerXML, char **value);
```

Visual Basic

```
Declare Function SK_XmlNodeGetValueString(ByVal flags As Long, ByVal xml As Long, ByVal xpath As String, ByVal includeInnerXML As Long, ByRef value As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`includeInnerXML`

When set to TRUE (1), all child nodes will be included in the returned string.

`value`

Reference/pointer to a string pointer, which will point to the string which contains the node's text value when the call succeeds. The string pointer must be initialized to NULL (0) prior to calling this function.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.
SK_ResultCode.SK_ERROR_XML_PARSER_FAILED	The XML parser encountered an error.
SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

SK_XmlNodeSetDocument Function

Imports an XML document under the node in the specified XML document.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeSetDocument(int flags, SK_XmlDoc xml, const char *xpath, SK_XmlDoc subDoc);
```

Visual Basic

```
Declare Function SK_XmlNodeSetDocument(ByVal flags As Long, ByVal xml As Long, ByVal xpath As String, ByVal subDoc As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`subDoc`

The XML document to import.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlNodeSetValueDateTimeString Function

Sets a node's text value to a given date-time string.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeSetValueDateTimeString(int flags, SK_XmlDoc xml, const char *xpath,
const char *value);
```

Visual Basic

```
Declare Function SK_XmlNodeSetValueDateTimeString(ByVal flags As Long, ByVal xml As Long,
ByVal xpath As String, ByVal value As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`value`

The node's new, date-time string value.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlNodeSetValueDouble Function

Sets a node to a given double floating point value.

Remarks

Important Note

When using this function in an application which uses vfork on POSIX-compatible platforms other than Windows and OS X, you should consider using the **SK_FLAGS_STRING_CULTURE_USE_CURRENT** flag to avoid unexpected behavior.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeSetValueDouble(int flags, SK_XmlDoc xml, const char *xpath, double value);
```

Visual Basic

```
Declare Function SK_XmlNodeSetValueDouble(ByVal flags As Long, ByVal xml As Long, ByVal xpath As String, ByVal value As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`value`

The node's new, double (floating-point) value.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description

SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlNodeSetValueInt Function

Sets a node to a given integer value.

Remarks

Important Note

When using this function in an application which uses vfork on POSIX-compatible platforms other than Windows and OS X, you should consider using the **SK_FLAGS_STRING_CULTURE_USE_CURRENT** flag to avoid unexpected behavior.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeSetValueInt(int flags, SK_XmlDoc xml, const char *xpath, int value);
```

Visual Basic

```
Declare Function SK_XmlNodeSetValueInt(ByVal flags As Long, ByVal xml As Long, ByVal xpath As String, ByVal value As Long) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`value`

The node's new, integer value.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.

SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_XmlNodeSetValueString Function

Sets a node's text value to a given string.

Syntax

C/C++

```
SK_ResultCode SK_XmlNodeSetValueString(int flags, SK_XmlDoc xml, const char *xpath, const char *value);
```

Visual Basic

```
Declare Function SK_XmlNodeSetValueString(ByVal flags As Long, ByVal xml As Long, ByVal xpath As String, ByVal value As String) As Long
```

Arguments

`flags`

Any **flags** passed into this function-call.

`xml`

The **SK_XmlDoc** handle.

`xpath`

XPath specifying the location of the node in the XML document. See the **License File Schema** topic for examples.

`value`

The node's new string value.

Returns

All possible return values are included in the **SK_ResultCode** enumeration. Return codes to expect include:

Result Code	Description
SK_ResultCode.SK_ERROR_NONE	No error.
SK_ResultCode.SK_ERROR_INVALID_DATA	The presence of invalid data has been detected.
SK_ResultCode.SK_ERROR_XML_NODE_MISSING	The requested XML node could not be found.
SK_ResultCode.SK_ERROR_INVALID_ARGUMENTS	Some or all of the arguments are invalid.

SK_ResultCode.SK_ERROR_MEMORY_ALLOCATION	Memory could not be allocated.

Types

Name	Description
SK_ApiContext	Handle to a SoftwareKey API Context.
SK_XmlDoc	Handle to an XML document in memory.

SK_ApiContext Type

Handle to a SoftwareKey API Context.

Remarks

Important Note

The API Context may only represent a single License File. When working with multiple License Files in a single application, one API Context will need to be created for each License File.

Important Note

When you have finished working with an API Context, it needs to be disposed to clear memory properly. The **SK_ApiContextDispose** function is recommended for this purpose.

Syntax

C/C++

```
typedef void *SK_ApiContext;
```

Visual Basic

```
Dim SK_ApiContext As Long
```

SK_XmlDoc Type

Handle to an XML document in memory.

Remarks

Important Note

When you have finished working with an XML document, it needs to be disposed to clear memory properly. The **SK_XmlDocumentDispose** function is recommended for this purpose.

Syntax

C/C++

```
typedef void *SK_XmlDoc;
```

Visual Basic

```
Dim SK_XmlDoc As Long
```

Visual Studio Integrated Help

If you are using the PLUSManaged and/or the PLUSManagedGui libraries in your application, the PLUSManaged API Reference is usually installed in Visual Studio (2010 or later) for you automatically when you install Protection PLUS 5 SDK. Likewise, this content is uninstalled for you automatically when uninstalling Protection PLUS 5 SDK.

This documentation format affords you the convenience of simply clicking on any class, property, or method from PLUSManaged or PLUSManagedGui and pressing F1 on your keyboard to see documentation on that specific entity. There may, however, be situations where this help was not installed for you already. One example is when the version of Visual Studio you are using was installed after you installed Protection PLUS 5 SDK.

Visual Studio 2010

Installing Content

- To install help content in Visual Studio 2010, begin by opening Visual Studio 2010 and clicking the *Help/Manage Help Settings* menu item.
- Next, click the *Choose online or local help* option, and make sure *I want to use local help* is selected. Keep in mind that selecting local help as your preference means Visual Studio will try to show you locally installed help first. If the relevant help topics are not installed locally, a hyperlink will be shown to allow you to view the help content online.
- Click *Install content from disk*.
- Click the *Browse* button and browse to the Protection PLUS 5 SDK installation folder. This is typically in *C:\Program Files (x86)\SoftwareKey\Protection PLUS 5* for 64 bit versions of Windows, or *C:\Program Files\SoftwareKey\Protection PLUS 5* for 32 bit versions of Windows.
- Select the *HelpContentSetup.msha* file, click *OK*, and click *Next*.
- On the next screen, click the *Add* link next to where *PLUSManaged API Reference* is listed and click the *Update* button.
- Next, you will see a security alert screen notifying you that the content is digitally signed. The content will only install once you click *Yes* on this screen.

Uninstalling Content

- To uninstall help content in Visual Studio 2010, begin by opening Visual Studio 2010 and clicking the *Help/Manage Help Settings* menu item.
- Click the *Remove content* option.
- On the next screen, click the *Remove* link next to where *PLUSManaged API Reference* is listed and click the *Remove* button to uninstall the content.

Other IDEs and Older Versions of Visual Studio

Integrated documentation content is not available for older versions of Visual Studio (2008 or earlier), as these use a very different format for integrated help. If you are using an older version of Visual Studio, or if you are using a different IDE, the PLUSManaged API reference is **available online** and is also installed with Protection PLUS 5 SDK.