InstallKey

Version 5.7

Installation Key
And
Validation Framework

By
LomaCons

May 17, 2015

Table of Contents

Software License

LomaCons InstallKey software is licensed, not sold. If you do not agree to these terms you cannot utilize this software and you must destroy all copies.

Custom licensing agreements can be made by contacting LomaCons.

This license governs use of the accompanying software. If you use the software, you accept this license. If you do not accept the license, do not use the software. This license agreement is valid without the licensor's signature or the licensee's signature. It becomes effective upon the licensee's installation or use of the software.

1. Definitions

The "licensor" is LomaCons.

The "licensee" is the user of this software.

A "computer" is a single physical computer or single instance of a virtual machine.

2. Grant of Rights

The following grants are subject to the terms of this license, including the conditions and limitations in section 3.

(A) The licensor grants the licensee a non-exclusive, worldwide, royalty-free license to use the software.

(B) The licensee may install the software on no more than four computers. If the licensee has purchased the source code, the licensee may install the software or derivatives of the software on no more than twenty-five computers. The licensee may make copies of the software for backup purposes only.

(C) The licensor grants the licensee the right to distribute, under a license of the licensee's terms, only the original, unmodified, non-derivative copy of LomaCons.InstallKey.ClientSide.dll and LomaCons.InstallKey.Common.dll, only when included in the licensee's distribution to perform validation of the licensee's distribution. If the licensee has purchased the source code, derivatives of these assemblies, in a similar compiled dll form, may be distributed under a license of the licensee's terms, only when these distributed assembly files have been renamed and obfuscated. All other files or derivatives of other files may not be redistributed.

3. Conditions and Limitations

(A) The software is licensed "as-is." The licensee bears the risk of using it. The licensor gives no express warranties, guarantees or conditions. The licensee may have additional consumer rights under local laws which this license cannot change. To the extent permitted under local laws, the licensor excludes the implied warranties of merchantability, fitness for a particular purpose and non-infringement.

(B) The licensor has the right to terminate this license agreement and licensee's right to use this software upon any breach by licensee.

(C) The licensee agrees to remove, uninstall and destroy all copies of the Software upon termination of the License.

(D) The licensee cannot rent or lease the software. The licensee may permanently transfer this license, provided that the licensee uninstalls and retains no copies, that the licensee transfers all the software, upgrades, derivatives, media and documentation and that the transferee agrees to be bound by the terms of this license.

(E) This license does not grant the licensee rights to use the licensor's name, logo, or trademarks.

(F) All right, title, copyrights and interest in the software are owned exclusively by the licensor. The software is protected by copyright laws and international treaty provisions. The software is licensed to the licensee, not sold to the licensee. The licensor reserves all rights not otherwise expressly and specifically granted to the licensee in this license.

(G) If for any reason a court of competent jurisdiction finds any provision of this license, or any portion thereof, to be unenforceable, that provision of this license will be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this license will continue in full force and effect.

Other Licenses

Portions of the InstallKey web application make use of components from the ASP.Net Ajax Control Toolkit. This toolkit is included in the InstallKey distribution and is redistributed in its unmodified, binary form under the CodePlex New BSD License (BSD) The license below applies only to the AjaxControlToolkit.

1. Product Overview

The LomaCons InstallKey framework can be added to any .Net 4.0 or 4.5 based applications to provide secure server validated software licensing and product installation key validation with installation counting. InstallKey is functionally similar to the licensing and product activation scheme used by Microsoft to license their Office and Windows products where you must validate a license key against both the installation hardware and a licensing server to install and use the software. In addition to this basic functionality, InstallKey can handle many, many other scenarios and business cases.

Features

InstallKey has the following features.

- Uses 30 character installation keys made up of unambiguous capital letters and numbers. This helps to make keys easy to read and enter by end users.
- Data entry key validation. This helps to catch most data entry errors before validating the installation key against the server.
- Installation key validation against a central server. Validating an installation key against a server should prove to be harder to bypass, as well as providing a mechanism to invalidate and revoke rogue keys.
- Unique key usage counting tied to the hardware where the software is installed. Key usage counting is done by comparing machine specific hardware identifiers, or other identifiers as defined by you.
- Written in Visual Studio ASP.Net C# and based on the .Net Framework 4.0. InstallKey can be used to validate any kind of .Net based application including Win Forms, ASP.Net Web Forms, ASP.Net MVC and other applications written in either C# or VB.Net. Supports any version of Visual Studio that supports .Net 4.0 or 4.5; version 2010 or newer.
- The database for the server component is Server 2008 R2 or later. The Express, Standard, or Enterprise editions are supported. The database is used to store the keys on the server. No database is required with your distributions.
- Client side validation against the server only needs to be done once. A tamper-proof surety is stored on the local system which is specific to the machine and key where the installation occurs. The surety can be verified again, at any time, without another round trip to the server. The surety can be stored as a file, a database record, or any other location where a string can be persisted.
- Custom data can be securely passed to and from the server during validation. Custom data is tamperproof and can be securely retrieved at any time after validation without a round trip to the server.
- Demo and Trial Keys that expire after a given date or timespan after first use. Keyless demos can also be implemented.
- Validation can be performed with or without an internet connection.
- Key Invalidation will allow a user to move a key to another installation without consuming another usage of the key.
- Additional interfaces and customization tools that allow InstallKey to be extended and customized to fit your needs and business requirements.

Overview

In the wild, validation and installation keys will probably not stop a determined hacker; however a good validation framework should keep your honest customers honest. Validation should be used to keep legitimate customers from grossly violating the software license agreement, and to prevent casual copying.

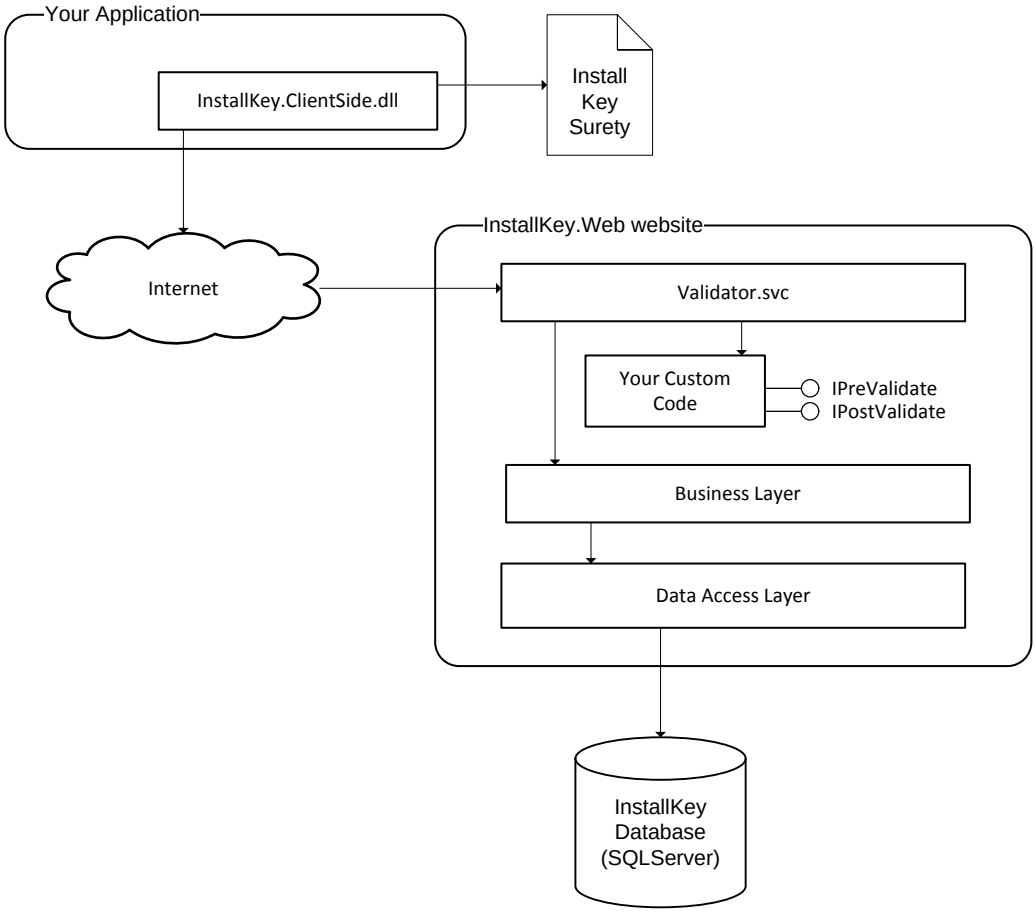
For example, the most common use for software licensing validation scheme is to prevent a customer from purchasing a single copy of your software and installing it on many, many more machines than agreed to.

To use InstallKey to add validation in your applications, you will need to reference the LomaCons.InstallKey.ClientSide.dll from your application. The objects and method calls in this assembly will perform validation of the installation key and return a result. If the key is valid, your application code will allow the user to continue.

The client side code performs validation against both the local machine hardware identifiers and a central server. Thus, you will maintain control over the validity of a customer's installation key at the time when the customer validates their key.

InstallKey Architecture

The diagram below shows the components and interactions of the InstallKey framework, and the descriptions that follow describe the components in more detail.



InstallKey Architecture

Your Application

Your application will reference the InstallKey Client Side dll. The Client Side dll encapsulates the key validation and surety verification as well as the web service call used to validate a key. Your application is left to determine when the validation occurs and how the validation results effects the functioning of your application based on the state of the surety.

InstallKey Surety
The surety is a tamper-proof xml file that contains the information necessary to validate the installation without having to contact the server. This is the same tamper-proof xml object that is sent to and from the server during key validation. The surety can also contain a collection of custom properties that you define and manage.

Internet and Validator.svc
Validator.svc is a .Net WCF web service that is hosted by IIS inside the InstallKey web application. This service does not require authentication for access and is called by the InstallKey.Client.dll whenever a key is validated against the server. When the key is validated, a tamper-proof surety is created and sent to this service for validation. Upon successful validation, the validated tamper-proof surety is returned to the client and stored locally for future validation.

InstallKey supports the most common customizations out of the box. Optionally you can add your own custom code to implement the IPreValidate and/or IPostValidate interfaces to provide additional customization at key validation time. These interfaces will allow you to customize the validation process, if needed.

InstallKey Database
All data and keys are maintained in a single SQL Server database. SQL Server 2008 R2 or later is required, and can be installed on the Express, Standard or Enterprise editions.

Caution: Do not lose or drop this database. Doing so will result in all of your customers keys being lost. Once lost, the exact same keysets and keys cannot be recreated. Always have a backup of your database!

InstallKey.Web Application
The InstallKey web application is the central point for both creating and validating installation keys. This web application contains both the web service, and visible web pages for maintaining your keysets, keys and properties.

The web application provides a simple user web based user interface on top of a three tiered internal architecture. Information a guides for using this application to manage your keysets and keys can be found in the following chapters.

2. Use Cases

The use cases that follow are based on this code snippet. This code snippet is an example of the code that you would call during the startup of your application to determine if the application is ready and validated for use.

```
private const string _keysetName = "...";
private const string _password = "...";
private const string _factor = "...";
private const string _validationCode = "...";
private const string  validationUrl = @"http://.../InstallKeyWeb/Validator.svc";

/// <summary>
/// Validates that the application is valdiated.  Will validate against
/// the server if the local surety file is not found or is not valid
/// </summary>
/// <returns>
/// true if the distribution is valid, false otherwise
/// </return>
public bool Validate() {
  ClientSideValidator validator;
  string key;
  EnumValidateResult result;

  // create an instance of the ClientSideValidator object
  validator = new ClientSideValidator(_keysetName, _password, _factor, _validationCode);
  validator.SetServiceAddress(_validationUrl);
  // check to see if the surety already exists and is valid
  if (validator.IsValidated) {
      return (true);  // already validated!
  }
  key = PromptUserForKey();
  // check to see if the user typed in the key correctly
  if (validator.DoesKeyMatchKeyset(key) == false)
      return (false);
  // validate the key against the server
  result = validator.ValidateAgainstServer(key);
  // check the result
  if (result == EnumValidateResult.Ok) {
      return (true);
  }
  return (false);
}
```

Performing Validation

Validation can be done at any time that is appropriate for your application, but typically it is performed during the first time your application is run, or at install time. The steps to accomplish this are as follows.

1. Create an instance of the ClientSideValidator object passing in the keyset information. Set the service address where the key will be validated against.
2. Check to see if the application is not already validated, by checking the IsValidated property.
3. Your code then prompts the user for their 30 character InstallKey. Use the DoesKeyMatchKeyset method to perform data entry validation and to verify that the key was entered properly by the user.
4. Call the ValidateAgainstServer method to validate the provided key value against the server.
5. If the validation was successful, the method returns true; otherwise, returns false.

What just happened here? What is going on inside these properties and methods?

The IsValidated property verified that the application was not already validated. (More about this in the next use case.) If the application was already validated, then the method returns true, and the application

is allowed to continue. The application does not need to ask the user for a key and does not need to contact the server for further validation.

The DoesKeyMatchKeyset method performs data entry validation by confirming that the key matches the keyset. This does not validate that the user has a valid key; rather, it can only confirm that the user typed it in correctly. It is possible, although unlikely, for a user to enter a key that matches the keyset, but is not a valid key. It is also possible, for a determined hacker to create a random key that matches the keyset. This is why the key must always be validated against the server.

The ValidateAgainstServer method performs a few activities. First, it retrieves information that is specific to the machine, including any network MAC addresses, Hard Drive serial numbers, Processor Identifiers and BIOS Identifiers. Next, the key and the identifiers are bundled into a surety and sent via a web service call to the business service web application for validation. Next, the web service validates the uniqueness of the identifiers relative to the key, and validates that the number of unique validations has not been exceeded. The resultant validated surety is then sent back to the client. Once back on the client the validated surety is persisted out to a local file. Along the way, the surety is validated to confirm that it was not tampered with or modified while traveling over the internet. (More about this local surety file in the next use case.)

What happens at this point is up to you. If the key is valid, you will typically allow the application to run. If the key is not valid, the application might terminate, or possibly run in a reduced functionality mode. How your application behaves when validated or not validated is up to you.

Validation on Subsequent Runs

After the application has been initially validated against the server, a surety file will have been created and stored on the local machine. The surety is a small XML file that contains all the information collected during the initial validation, including the installation key and the machine specific identifiers.

Additionally, the surety can contain custom information in a property collection that is securely embedded in the surety. (See the section on sending custom data to/from the server for more information.)

The IsValidated property is used to validate that the application is already valid by comparing the information in the surety file and the current machine identifiers. The current machine specific information is found, and compared to the values stored in the surety file. If the values are largely different, the validation will fail and the IsValidated property will return false. If the values are the same, the IsValidated property will return true.

Notice that it is not necessary to validate against the server each time you want to verify that the validation has been successful. This is because the surety file is tamper-proof and includes the machine specific information at the time it was validated against the server. Therefore we can assume that the application is valid as long as the surety was not compromised, and the machine specific information has not changed.

Validation and the Validation Web Service URL

When the ClientSideValidator object validates an installation key against the server, it will connect to your predefined web service URL. This URL is a WCF based web service that is defined in the InstallKey web application config file. The service is hosted in IIS inside the InstallKey web application.

The ValidateAgainstServer method can be called any time that you want to refresh or revalidate the key against the server. As long as the unique identifiers are the same, the surety is refreshed and usage count will not be incremented.

Validation Result Codes

In the sample code snippet above, the returned result is compared against EnumValidationResult.Ok to indicate a Boolean result. This is acceptable in a simple scenario; however, you might want to use the

enumeration result to provide additional feedback to the user or other handling specific to your application. Below is the description of the result codes, what they are indicative of, and why they might occur.

EnumValidationResult.Ok
This value indicates that the validation against the server was successful and the identifiers were successfully stored on the server. If this is the first validation using this set of unique identifiers, then the identifiers will not be found on the server and the usage count was increased. If matching identifiers were found on the server, the usage count will not be increased.

EnumValidationResult.Violation
This value indicates that the key was found on the server; however, this attempt to validate failed because this validation is a new unique validation that would exceed the limit allowed for the key. This unique usage of this key would have exceeded the maximum usages allowed for the key.

EnumValidationResult.InvalidKey
This value indicates that the key was not found on the server and does not exist on the server. Typically this happens because it was deleted on the server.

EnumValidationResult.Revoked
This value indicates that the key was found on the server; however, the key was revoked and is no longer valid.

EnumValidationResult.PreValidateFailure
This value indicates that the key was found on the server; however the custom pre validation code did not allow the key validation to succeed. This value is only possible when a custom pre validate handler has been implemented. See the chapter on Customizing the Validation Process for more information.

3. Thwarting Attempts to Bypass the Validation

There are some common scenarios that your users may attempt to try to bypass the key validation and surety verification process. Because the surety file is tamper-proof and the contents of the surety are tied to the specific machine where it was validated and installed, InstallKey can successfully prevent these attempts.

Casual Copying

Casual copying occurs when a user copies your application to another machine, either intentionally or unknowingly. Typically the user will copy the entire set of application files including the surety.

Attempting to copy the surety file to another machine will cause the validation to fail on the other machine, but it will continue to work on the original source machine. This is because the machine specific information on the second machine will not match the information stored in the surety file. In this case the IsValidated property will return true on the original machine, but will return false on the second machine.

Modifying the Surety File

A user may attempt to bypass the surety, or change its contents, by modifying the surety file itself.

To prevent this, the surety file contains a validation code that is used to validate and verify that the surety file content has not been tampered with. Attempting to modify the surety file contents or the validation code will cause the surety file to become invalid. In this case, the IsValidated property will return false.

Brute Force Attacks

Every system can be bypassed by a brute force attack where a rogue user tries repeatedly to enter valid keys. InstallKey is no exception; however, precautions can be taken to limit this in both your code and by InstallKey.

First, even if a rogue user entered a random key that that matches the keyset, your code should always be written such that it still has to pass the server validation. The total number of unique keys is so large that the probability of the same key being generated by the server and randomly selected by the rogue user is infinitesimally small, the key most likely will not exist on the server and the validation will fail.

Second, your application can prevent a brute force attack by limiting the number of times a user can attempt to enter an invalid key. Your code can be written such that multiple attempts to validate with an invalid key will cause the application to terminate, thus making it quite inconvenient to enter many keys repeatedly.

Third, even if a rogue user manages to enter a valid key that also exists on the server, the key can be revoked at any time to prevent further usage attempts.

Revoking Keys

Keys are intended to be used multiple times by a single customer up to the violation limit. There's not much you can do to prevent a key from being shared or getting out in the wild. But InstallKey does provide a key revocation mechanism to handle this scenario.

By revoking the key, you effectively disable the key at the server. All further attempts to validate a revoked key against the server will always fail, regardless of the number of times the key has been used. What this does is prevent any future validations with that specific key to succeed.

4. InstallKey Installation Prerequisites

The InstallKey web application requires the installation and configuration of Microsoft Windows components and features. These features are typically not installed or activated when the Windows Operating System is installed. However, they can be easily added using the Add/Remove Programs applet or Server Manager Tools in the Windows Control Panel.

- Internet Information Services
- .Net Framework 4.0 or later
- SQL Server 2008R2 or later; Express or greater edition

Specific and required components of these features are defined below, along with any notes about any differences for the different major versions of the Windows operating system. In general, if you are already developing applications with Visual Studio, these components are likely already installed.

Please note that IIS is not available in the Starter, Home or Basic editions of Windows. Because of this, the InstallKey web server components cannot be installed on these editions. Also note that this limitation applies only to the InstallKey Server. The LomaCons.InstallKey.ClientSide.dll and LomaCons.InstallKey.Common.dll files that you include with your application can run on any edition of Windows.

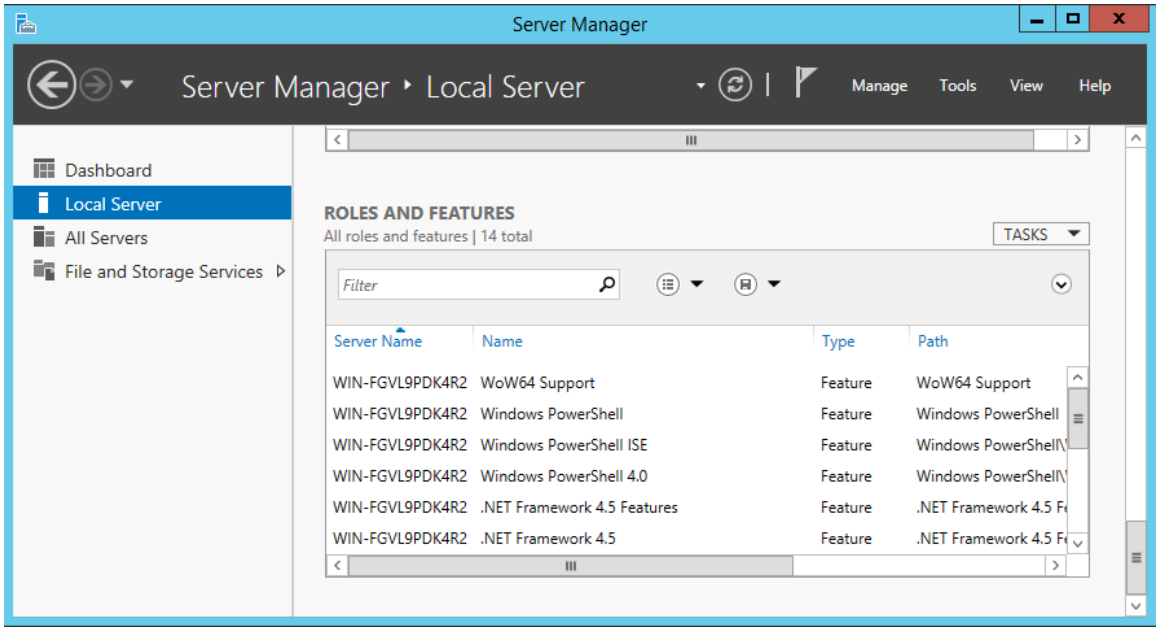
Internet Information Services

IIS must be installed on your web server prior to installing InstallKey. IIS is included with various editions of the Windows desktop and server operating systems, but is not installed by default. It can be added any time from the Add/Remove Programs or Server Roles applet in the Windows Control Panel.

Once IIS is installed, installation and support for ASP.Net 4.x features in IIS must also be enabled.

Installing IIS on Windows Server 2012 R2
The IIS settings can be found in Server Manager. If you do not yet have IIS installed, follow these steps.

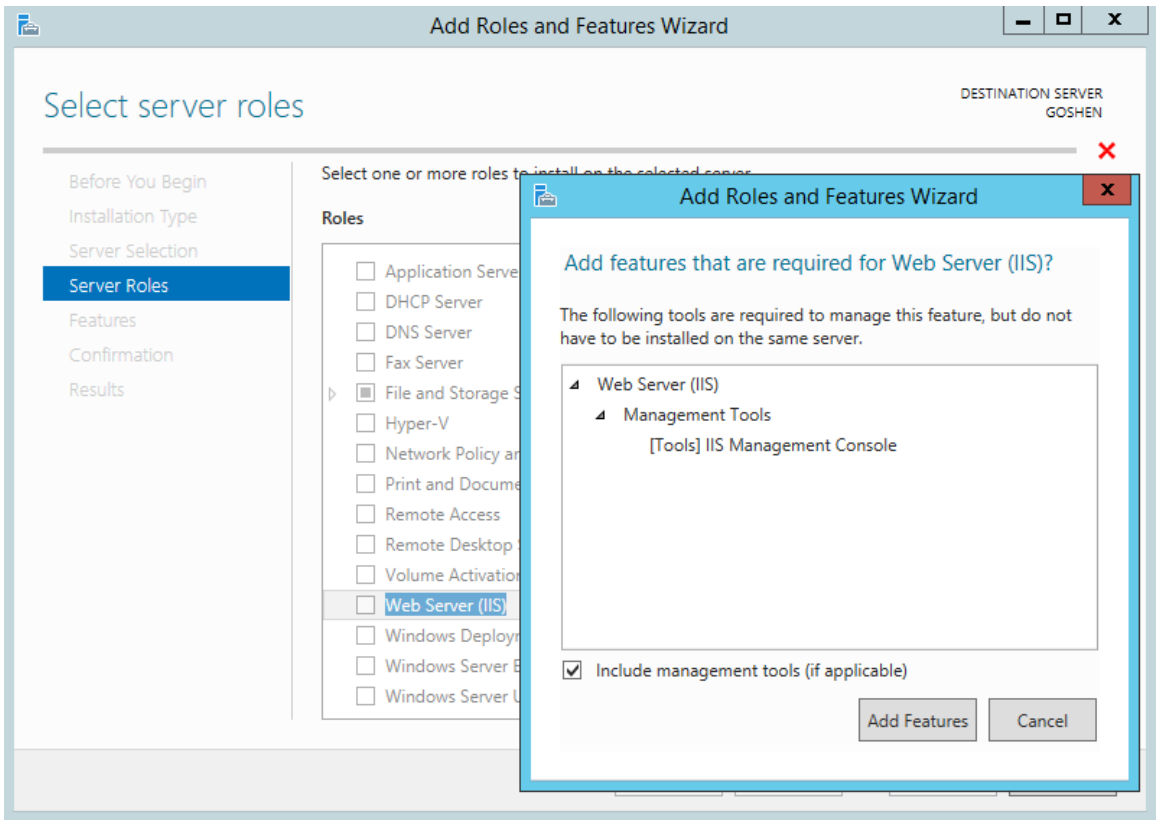
1. Open Control Panel and select Administrative Tools.
2. Open Server Manager, and select Local Server.
3. Scroll down to the Roles and Features section.



Server Manager – Roles and Features

4. From the Roles and Features Task drop down, choose Add Roles and Features. This will start the Add Roles and Features Wizard.

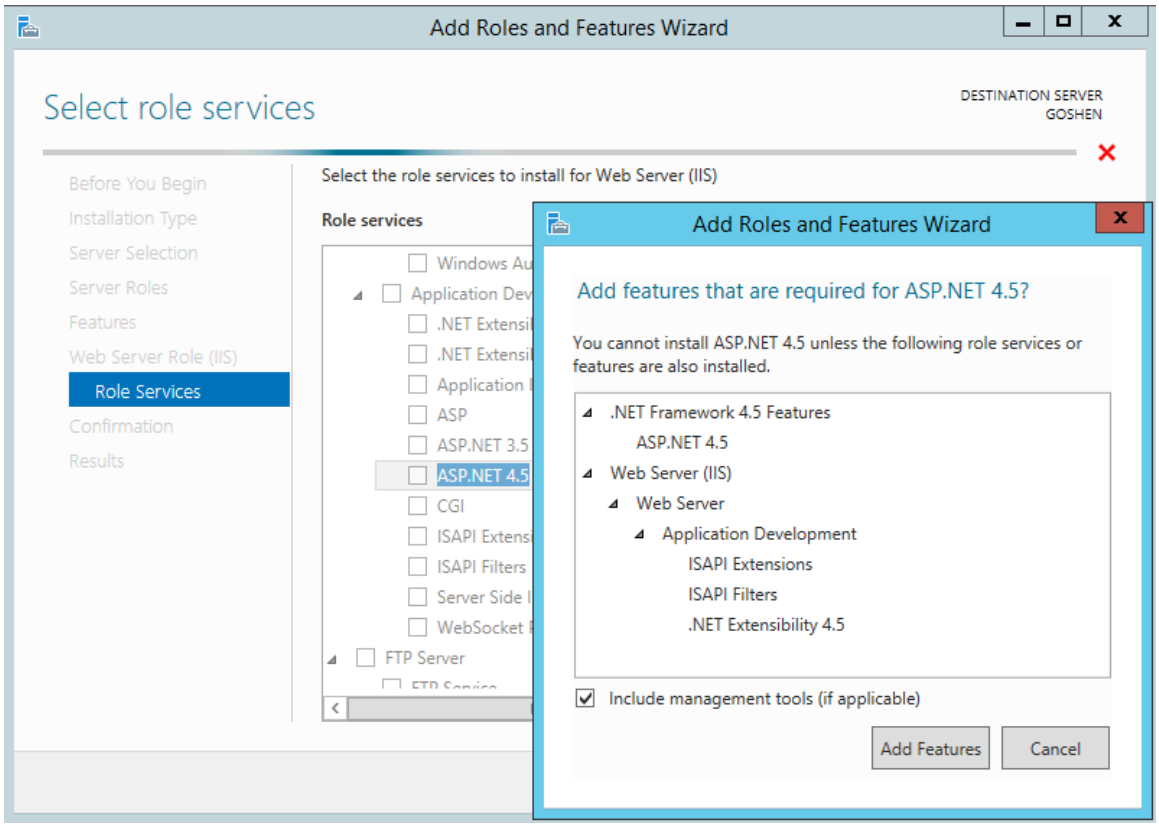
5. In the Server Roles list choose Web Server (IIS). Approve the addition of any required features.



Server Manager Roles

6. Click the Next button to advance the wizard to the Web Server Role Services.

7. Windows will have preselected the minimum required IIS settings. In addition to the preselected settings, you also need to select the Application Development | ASP.Net 4.5 checkbox. Selecting this check box will also include other required role services. Select the checkbox, and accept the additional role services.



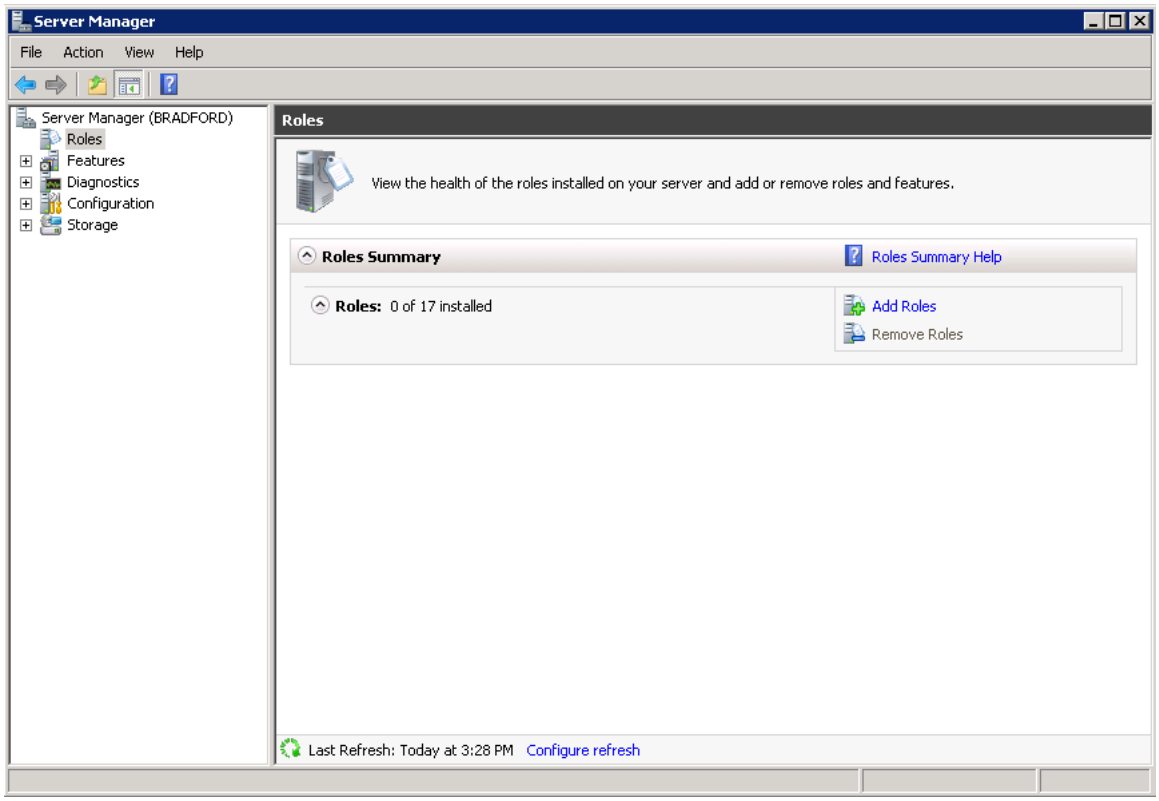
IIS Role Services with ASP.Net Selected

8. Click the Next Button to complete the wizard and install IIS.

Installing IIS on Windows Server 2008 R2

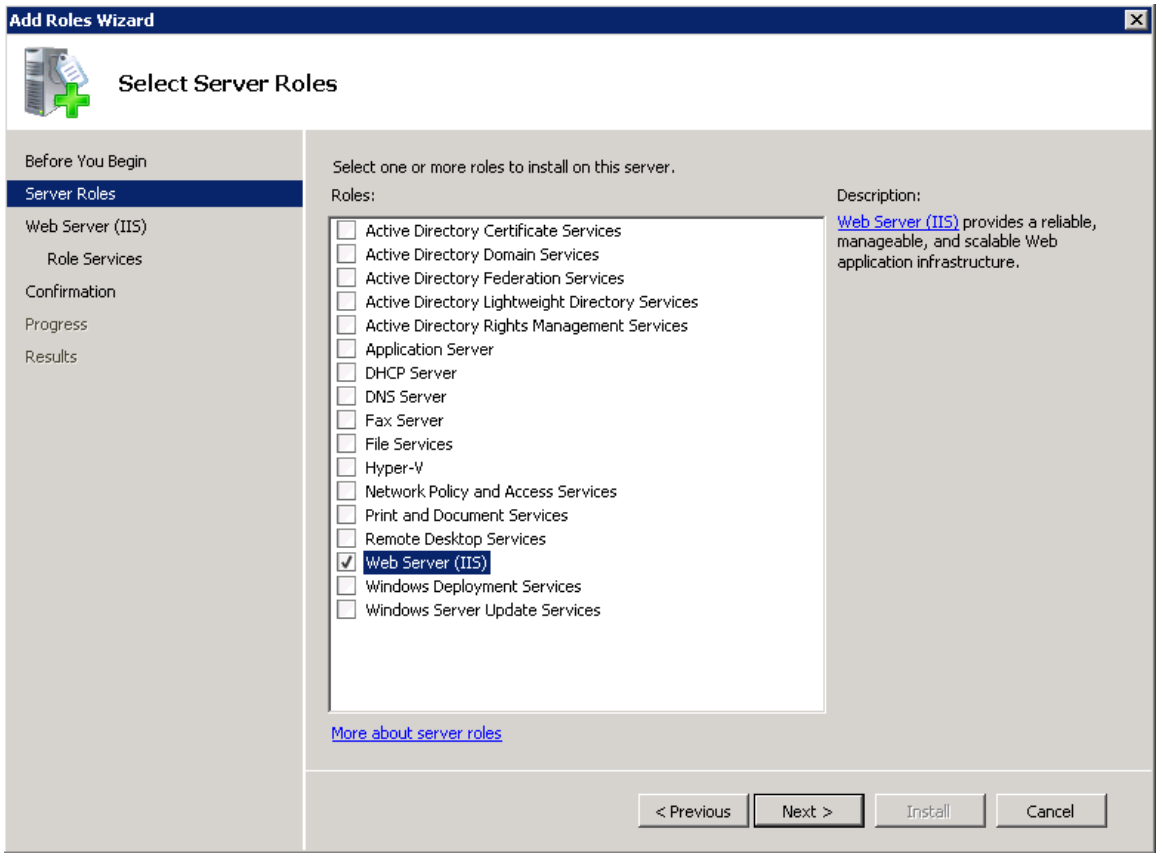
The IIS settings can be found in Server Manager. If you do not yet have IIS installed, follow these steps.

1. Open Control Panel and select Administrative Tools.
2. Open Server Manager and select the Roles Node in navigation tree.
3. Click the Add Roles Link to start the Add Roles Wizard.



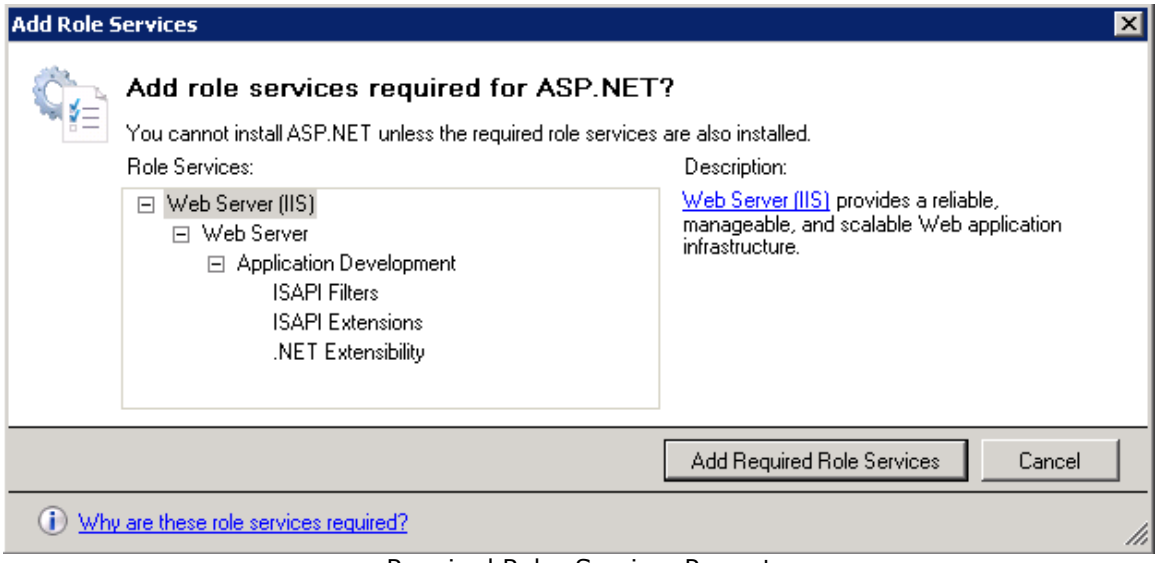
Server Manager - Roles

4. In the Server Roles list choose Web Server (IIS)

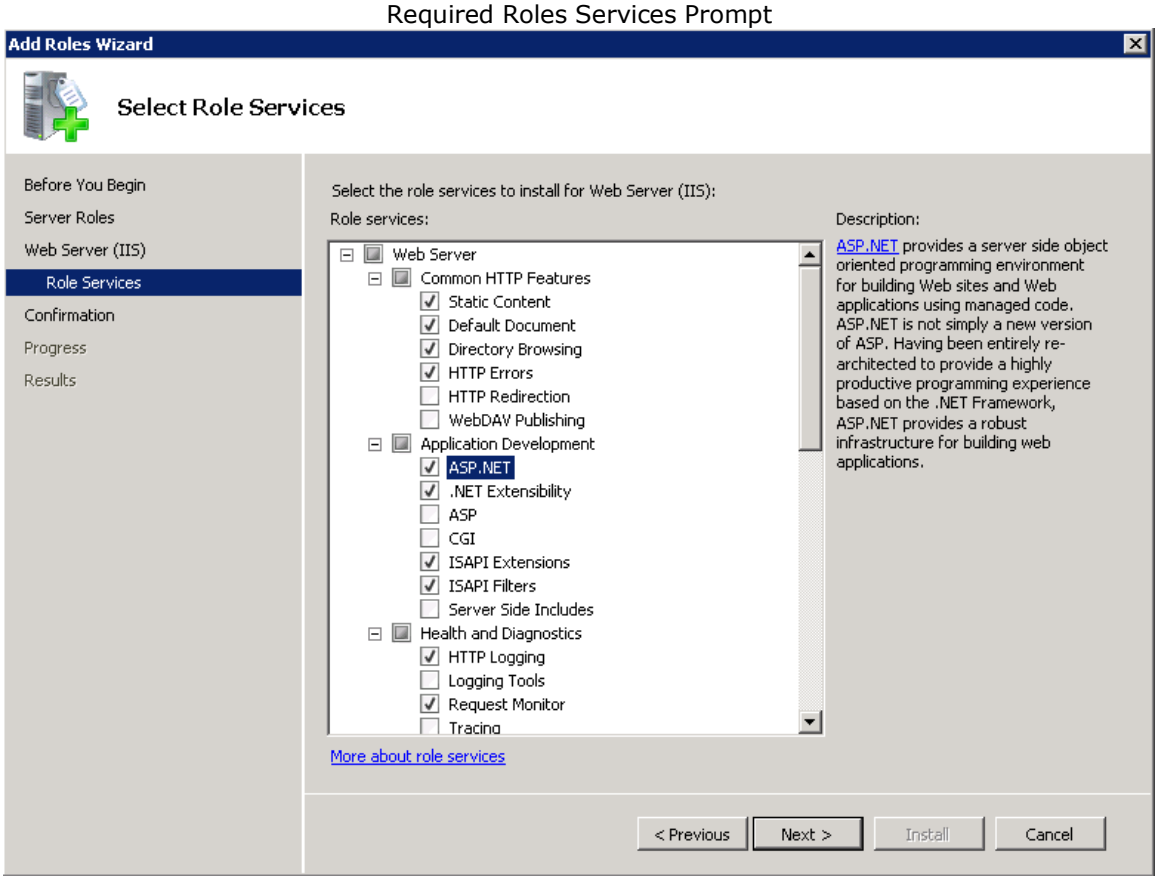


Select Web Server (IIS) Role

5. Click Next to see the Role Services list. Windows will have preselected the minimum required IIS settings. In addition to the preselected settings, you also need to select the Application Development | ASP.Net checkbox. Selecting this check box will also include other required role services. Select the checkbox, and accept the additional role services.



Required Roles Services Prompt



IIS Role Services with ASP.Net Selected

6. Click the Next Button to complete the wizard and install IIS.

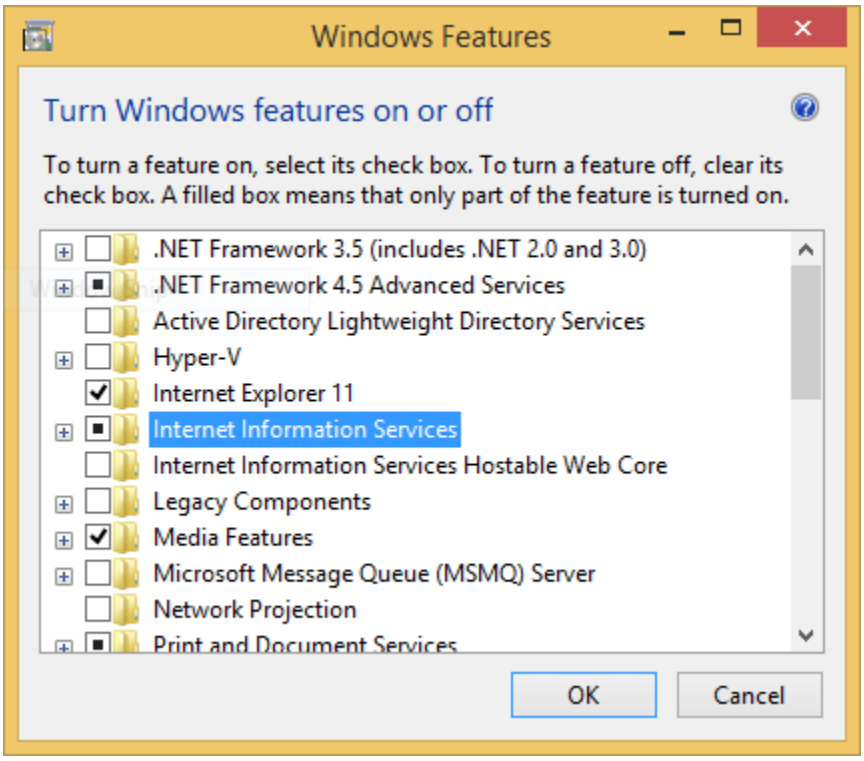
Installing IIS on Windows 8.1

The IIS settings can be found in Windows Control Panel.

1. Switch to the desktop

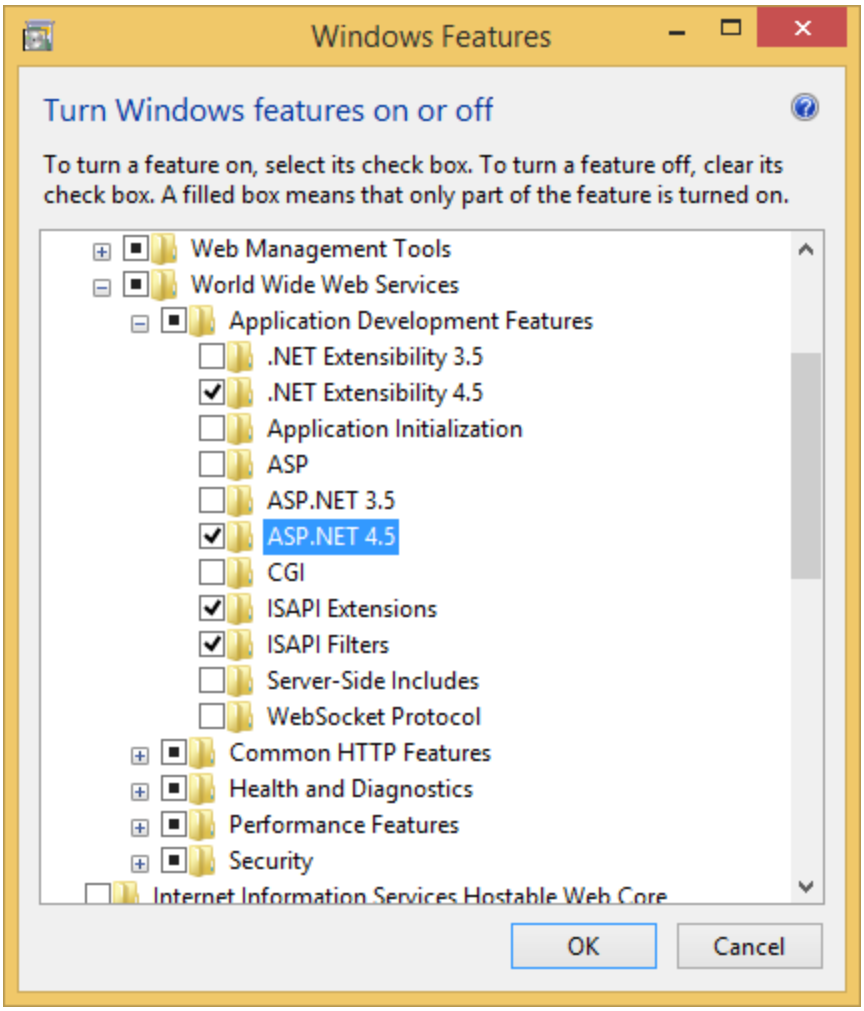
1. Open the Windows Control Panel

2. Open the Programs and Features applet
3. Click the Turn Windows Features on or off link
4. In the Windows Features dialog, enable Internet Information Services. This will enable the base installation of IIS.



Enabling Internet Information Services

5. Expand the Internet Information Services node.
6. Expand the Application Development Features node.
7. Select the ASP.Net 4.5 option. This will auto select a few other required IIS features.



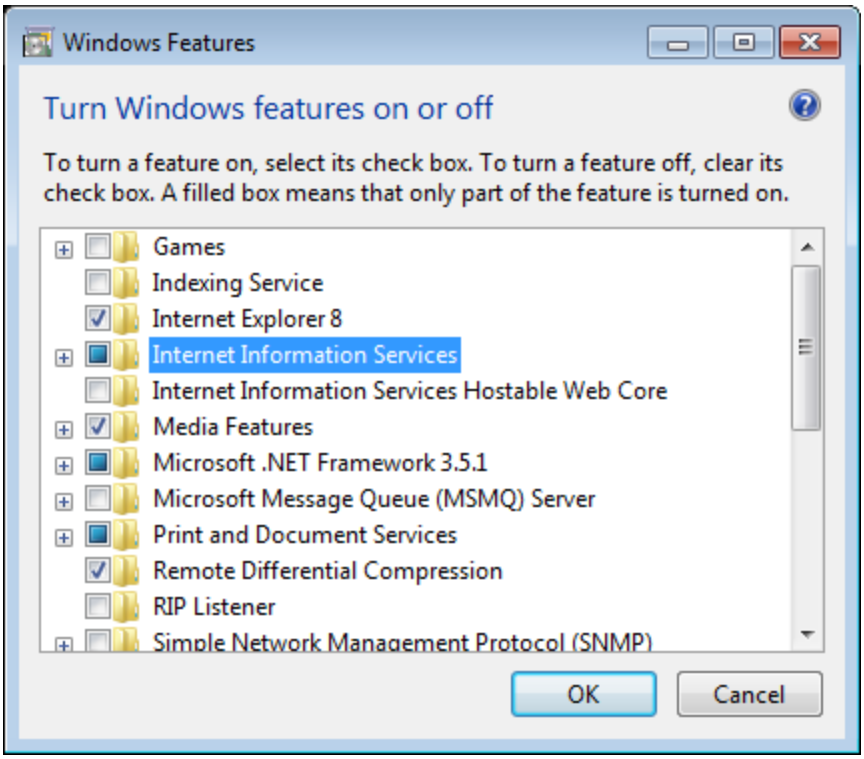
Enabling ASP.Net Functionality in IIS

8. Click Ok to save the settings and install IIS.

Installing IIS on Windows 7

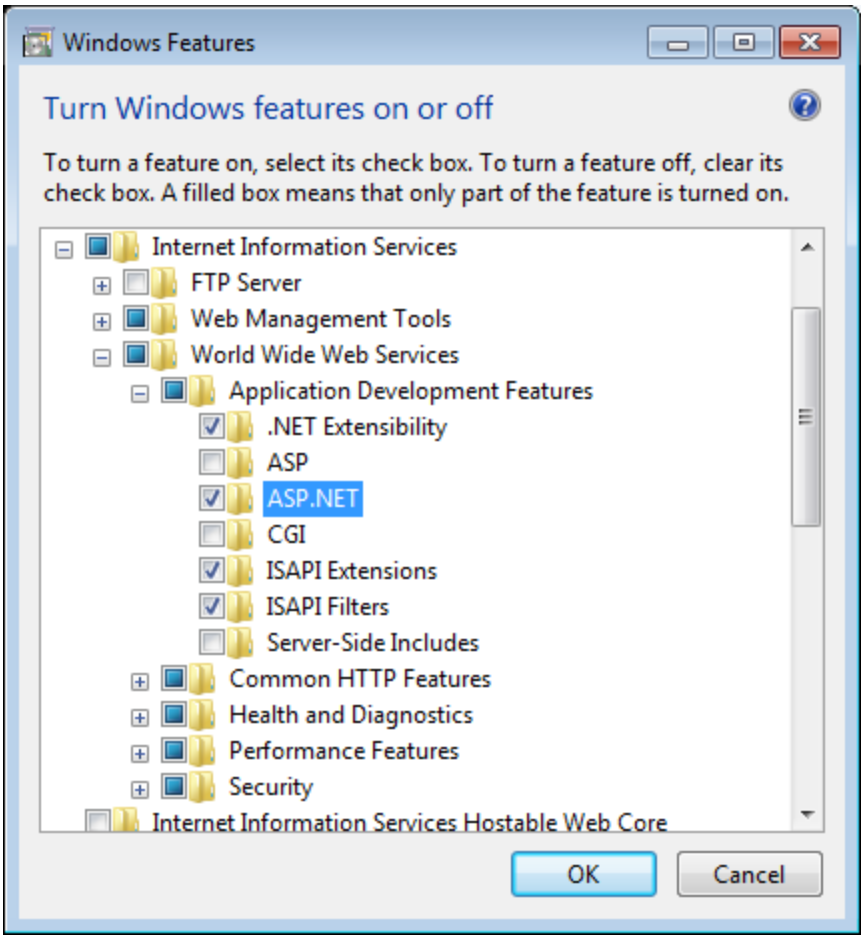
The IIS settings can be found in Windows Control Panel.

1. Open the Windows Control Panel
2. Open the Programs and Features applet
3. Click the Turn Windows Features on or off link
4. In the Windows Features dialog, enable Internet Information Services. This will enable the base installation of IIS.



Enabling Internet Information Services

5. Expand the Internet Information Services node.
6. Expand the Application Development Features node.
7. Select the ASP.Net option. This will auto select a few other required IIS features.



Enabling ASP.Net Functionality in IIS

8. Click Ok to save the settings and install IIS.

The .Net Framework 4.x and WCF HTTP Activation

The Windows 7 and Windows 2008R2 operating systems do not include the .Net framework 4.0, and it will need to be installed separately if not already installed on the machine where you are installing the InstallKey server. This framework is available as a free download from Microsoft.

Conversely, Windows 8.1 and Windows Server 2012R2 include .Net 4.x as part of the operating system installation. However, on these operating systems additional features of the framework must be enabled for the proper functioning of the InstallKey WCF Validation web service.

Because InstallKey uses WCF over HTTP for confirming validations, the HTTP Activation must be enabled in some Windows operating systems. Please confirm that the .Net 4.x WCF Services has the HTTP Activation enabled. This setting is required for InstallKey to function properly and can be found inside control panel in Windows. The location of this setting is in a slightly different location in each version of Windows.

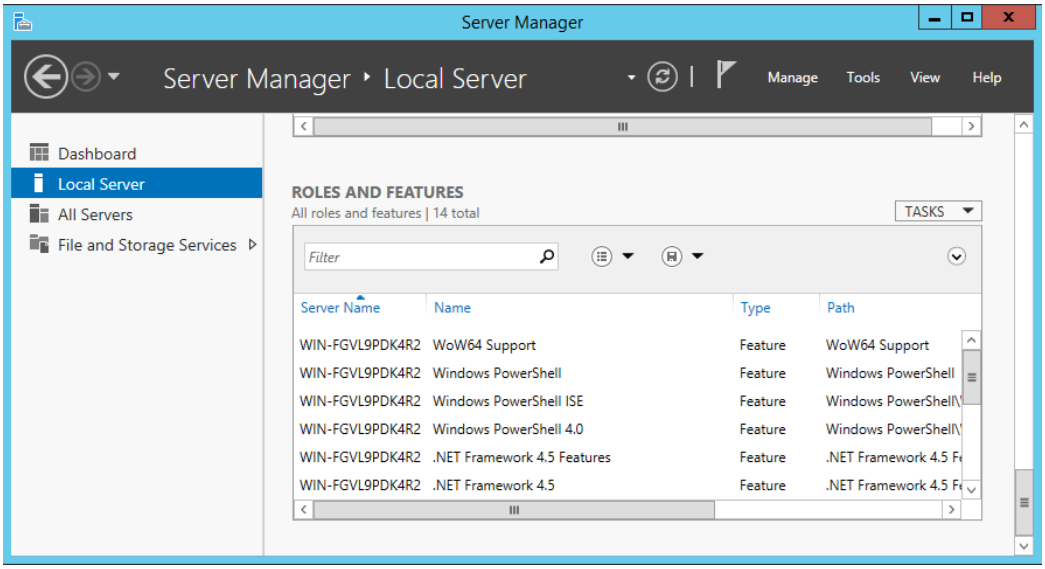
Note that version 4.5 or newer of the .Net Framework is backwards compatible with the .Net Framework 4.0. In the diagrams and notes below you may see references to version 4.5, however InstallKey will function with either version of the framework.

Windows Server 2012 R2

The WCF HTTP Activation setting can be found in Server Manager. If you do not yet have HTTP Activation enabled, follow these steps.

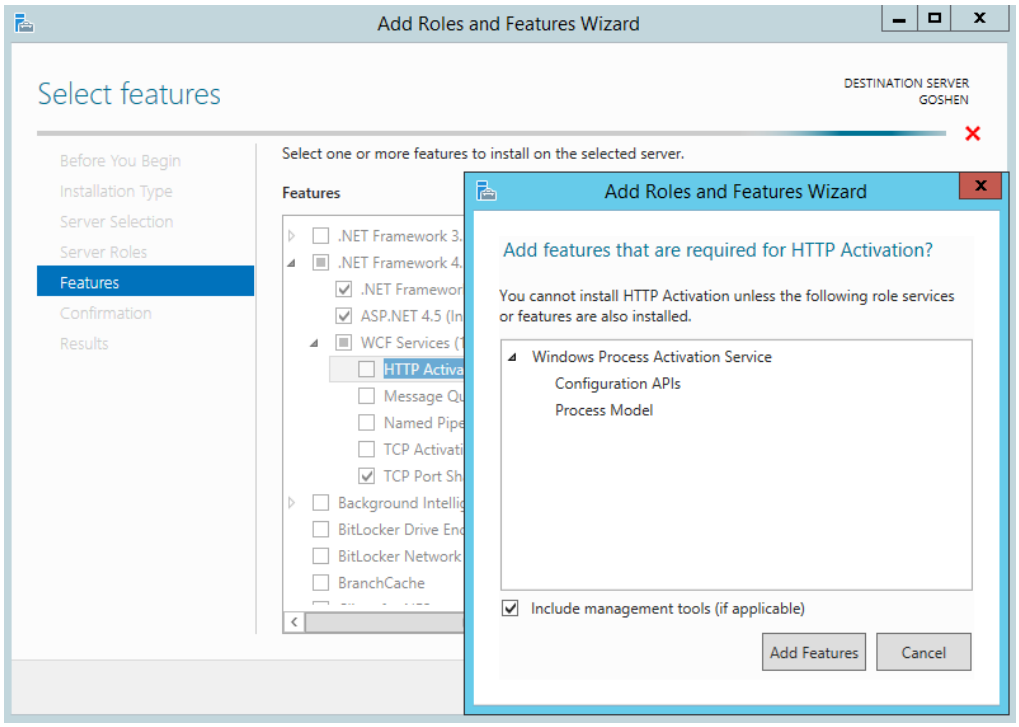
1. Open Control Panel and select Administrative Tools.

2. Open Server Manager, and select Local Server.
3. Scroll down to the Roles and Features section.



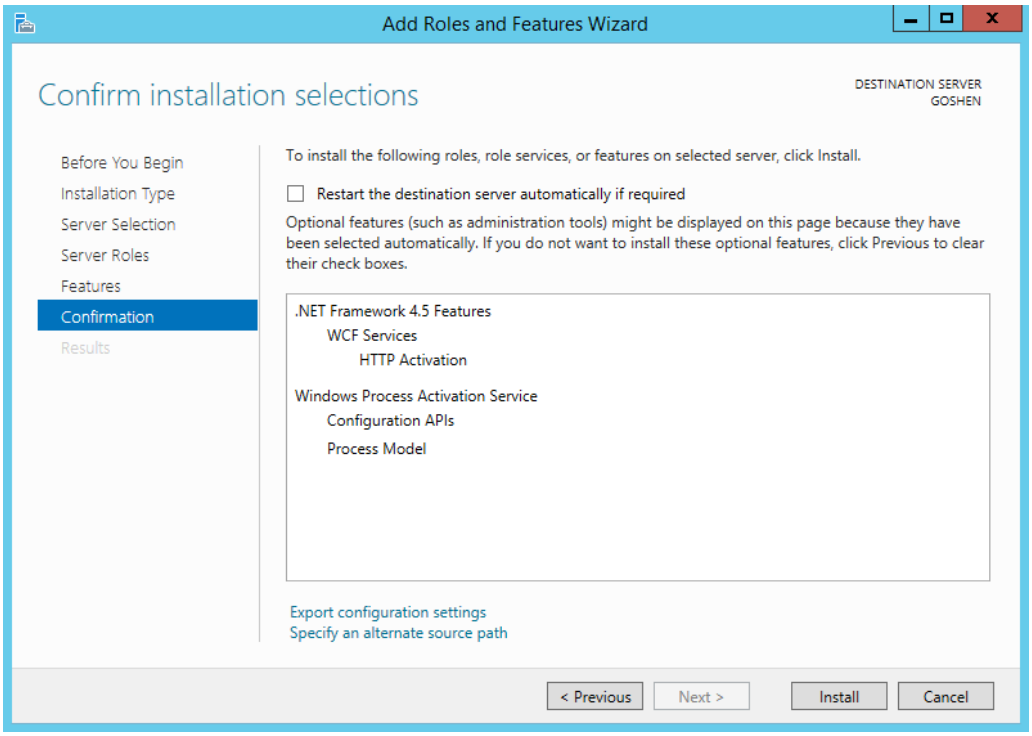
Server Manager – Roles and Features

4. From the Roles and Features Task drop down, choose Add Roles and Features. This will start the Add Roles and Features Wizard.
5. Click Next to Advance to the Features List.
6. In the Features list expand the .Net Framework 4.5 Features node. Under that, expand the WCF Services node.
7. Check the checkbox for HTTP Activation. If the Add required features dialog appears, accept the addition of the required additional features.



Server Features

8. Click the Next Button to complete the wizard to enable .Net 4.5 HTTP Activation.



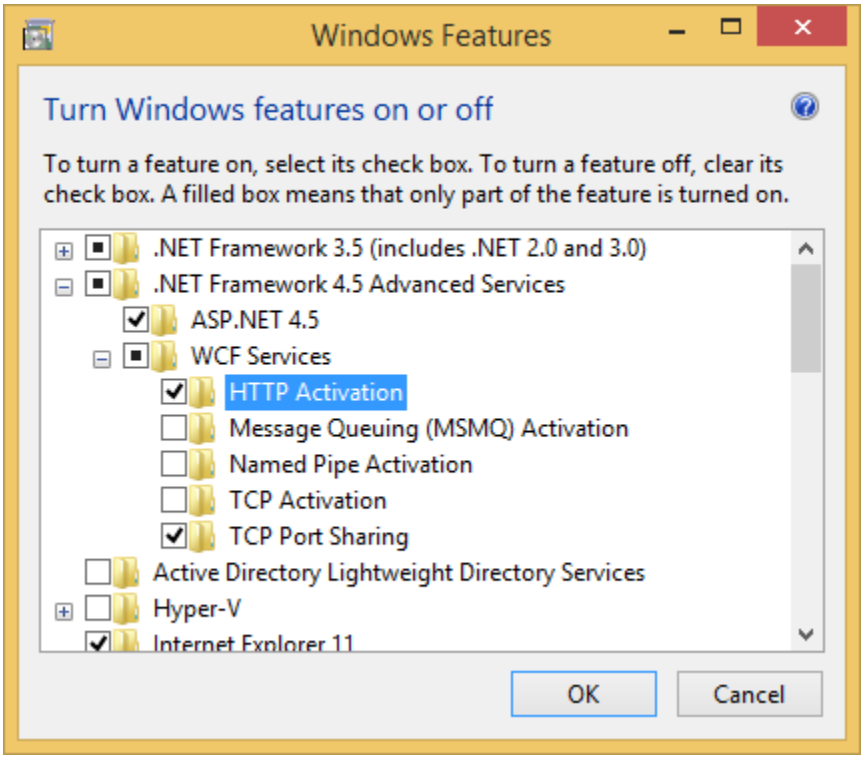
Confirmation To Install and Activate WCF HTTP Activation

Windows Server 2008 R2

The .Net Framework 4.0 is not included with Windows 2008 R2 but it can be downloaded from Microsoft's website. If you are installing both IIS and .Net 4.x, install IIS first, then the .Net Framework. It is recommended that you download and install the most recent version of the 4.5 framework

Windows 8.1
The HTTP Activation setting can be found in Windows Control Panel.

1. Switch to the desktop
2. Open the Windows Control Panel
3. Open the Programs and Features applet
4. Click the Turn Windows Features on or off link
5. In the Windows Features dialog, enable HTTP Activation and click OK



Windows 8.1 Features - .Net Framework HTTP Activation

Windows 7
The .Net Framework 4.0 is not included with Windows 7 but it can be downloaded from Microsoft's website. If you are installing both IIS and .Net 4.x, install IIS first, then the .Net Framework. It is recommended that you download and install the most recent version of the 4.5 framework.

SQL Server

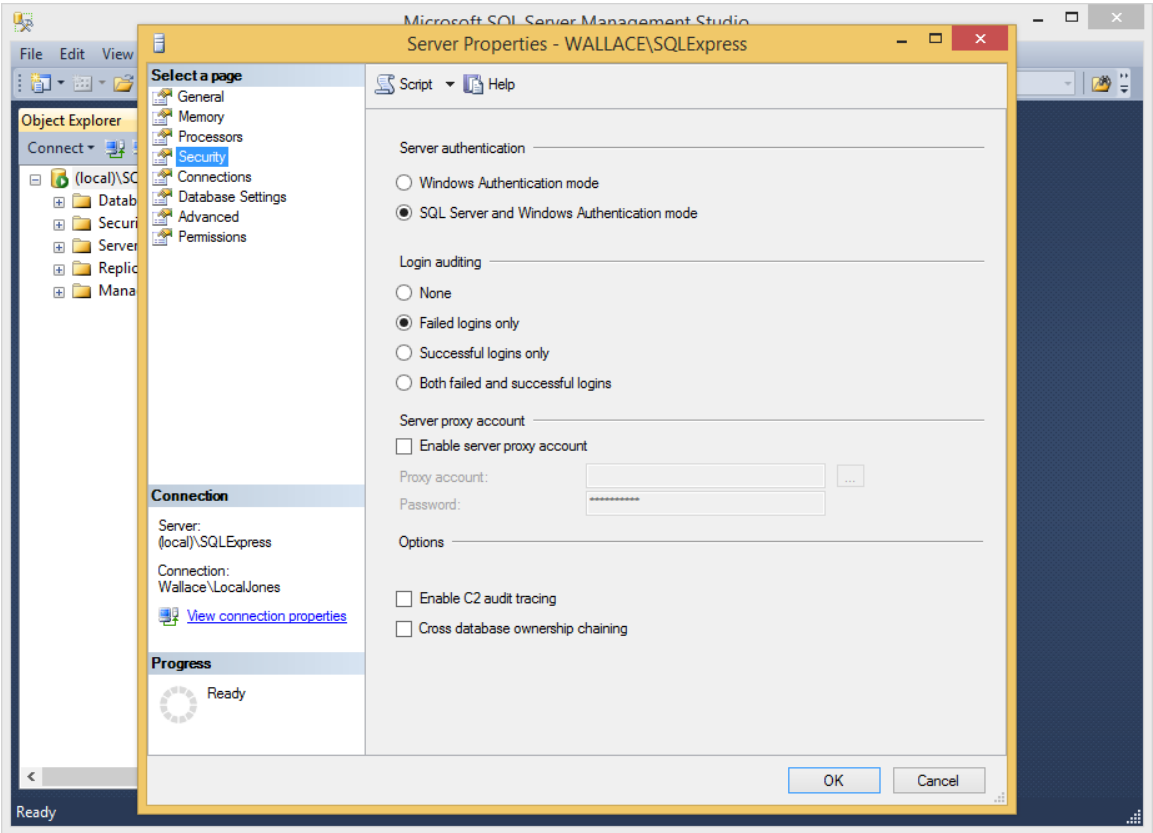
The InstallKey web application must have access to an Enterprise, Standard or Express edition of SQL Server 2008 R2 SP2 or newer. The SQL Server instance does not have to be on the same machine. Administrative or sa rights to the SQL Server instance will be needed only to create the database and database login.

Once the database has been created, a login with minimal security access rights can be used for the connection to the database from the InstallKey application. A SQL script to create this login with minimal access is provided.

Mixed Mode Authentication
Often the easiest way to configure application level access to the database is by using a SQL login and password. This type of login in is stored securely in SQL Server.

If SQL Server was installed with the default authentication setting, then Mixed Mode will likely be disabled. To enable Mixed Mode after SQL Server has been installed, change the setting on the SQL Server instance properties.

1. Open SQL Server Management Studio and connect to your database server instance
2. Right click on the server instance and choose Properties
3. In the properties dialog, choose the Security page
4. Confirm that the Server Authentication setting is set to SQL Server and Windows Authentication mode.



Changing the SQL Server Authentication Mode

SQL Server Integrated Windows Authentication
Coming Soon! – Application access to SQL Server using Windows Authentication will require changes to the IIS Application Pool settings in addition to creating a corresponding windows login in SQL Server and changes to the connection string in the web.config file.

Documented support for this configuration will be added in a future release of InstallKey, however if you are knowledgeable about setting this up, it should be possible with this release of InstallKey.

5. InstallKey Installation

InstallKey consists of a few different components that work together to provide key creation, data storage and key validation. These components were created with 100% .Net managed code and common Microsoft development tools and technologies.

- The client side components are in two dlls, LomaCons.InstallKey.ClientSide.dll and LomaCons.InstallKey.Common.dll. These two dlls are referenced by your application to provide the validation functionality and may be redistributed only with your application.
- The server side component is a web application called InstallKeyWeb. This web application is the central point for managing and validating the keys. It contains both the key administration web pages and the validation service. The Web based UI allows you to administer the InstallKey server to create keysets, keys, revoke keys, change install count limits and more.
- A SQL Server database. This database holds all the information about the installation keys. You should always have a backup of this database. Dropping or losing this database will cause all of your keys to become invalid.
- Sample applications. The sample applications written in C# or VB.Net provide examples of how to call and use the client side code.

Installing InstallKey

Setup and configuration of InstallKey is straight forward and simple. If you have built and deployed web applications based upon the Microsoft .Net framework, you will have no trouble with the installation. Alternatively, if you have never installed a web application before, the setup is just as easy by following the guide and steps below.

InstallKey is written in 100% .Net managed code and follows Microsoft's guidelines for application design and deployment. There are no complex or mysterious setups or tools. In general, the process is to create the database, create the website in IIS, and configure the database connection string.

To begin, download the current release of InstallKey from http://www.lomacons.com. This zip file will contain a collection of zip file packages.

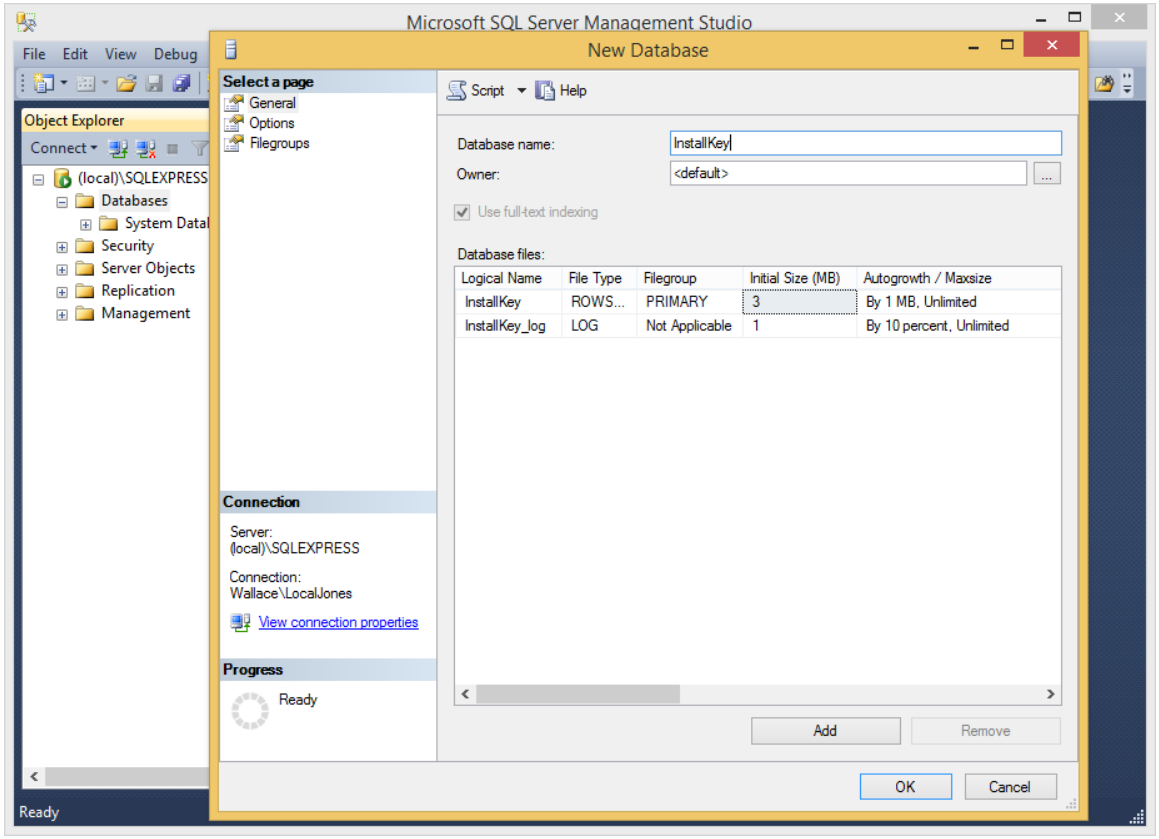
- Database.zip – REQUIRED – This zip file contains the database script files. These files will be run to create and update the database schema and fixed content. A database login script file is also included.
- InstallKeyWeb.zip – REQUIRED – This zip file contains the deployment files for the InstallKey web application. The application is already compiled, and the files will be copied to your IIS server file system.
- ClientSideBinaries.zip – REQUIRED – This zip file contains the LomaCons.InstalllKey.ClientSide.dll and LomaCons.InstallKey.Common.dll files. These are the two .Net assemblies that you will need to include as references in your application project file. You also will need to redistribute these files with your application. The client side libraries do not require any additional Windows features other than the .Net Framework 4.0 for normal client side operation.
- ClientSideTester.zip – This zip file contains optional content. This executable file can be used to validate a key against your server, can be used to confirm that you have installed your server properly and that you can connect to the validation service endpoint. The source code for this application is included in the samples.
- ClientSideSamples.zip – This zip file contains optional content. This zip file contains a number of Visual Studio projects written in C# or VB.Net that are real working samples that demonstrate how to use the client side objects, methods and properties.

- InstallKeyDevGuide.pdf – This pdf file contains full and complete documentation about InstallKey. (You are reading this file now.)
- QuickStartGuide.pdf – This pdf file contains condensed installation instructions.
- ReleaseNotes.pdf – This pdf file contains the full revision history of InstallKey from version 1.0 through the current release. This includes lists of new features, obsolete features, removed features, breaking changes and bug fixes.

Database Creation and Setup

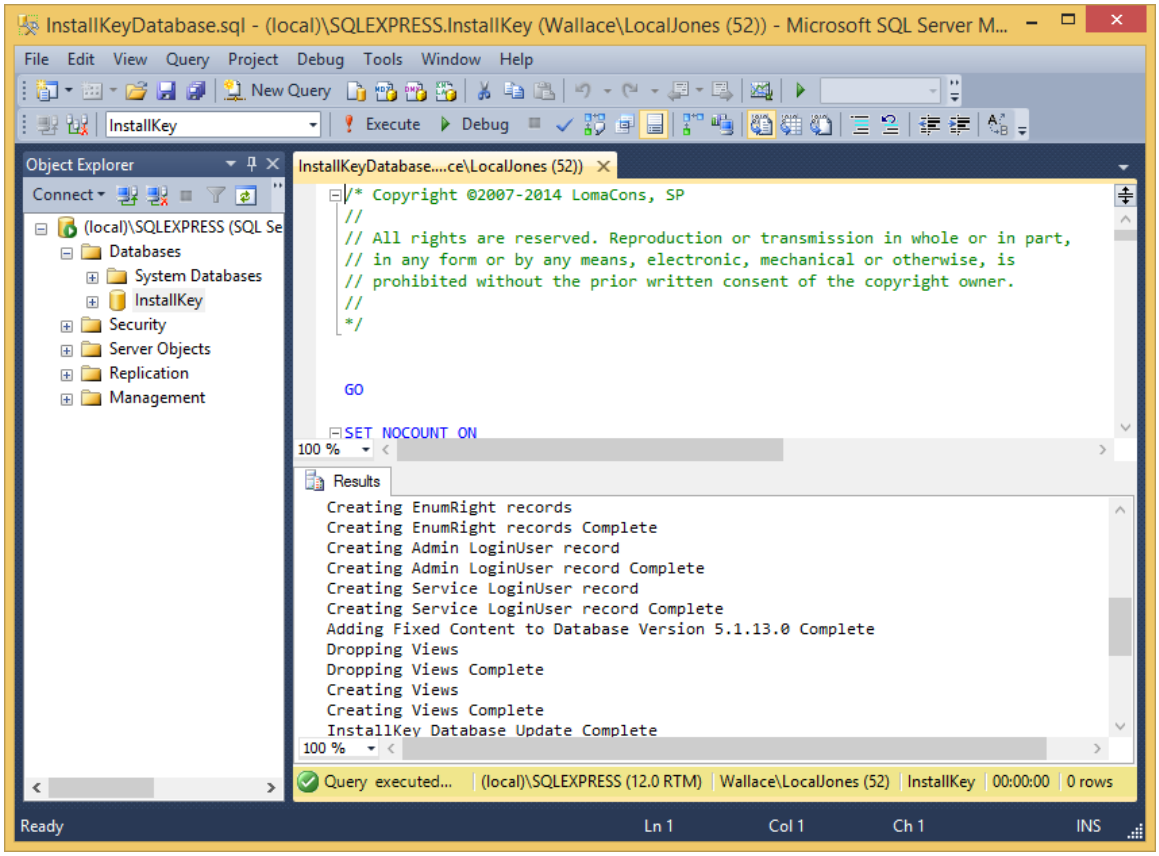
Follow the steps below using SQL Server Management Studio (SSMS) to create the InstallKey database.

1. Create a new empty database. Enter InstallKey for the database name, and leave the defaults for all other options and settings.



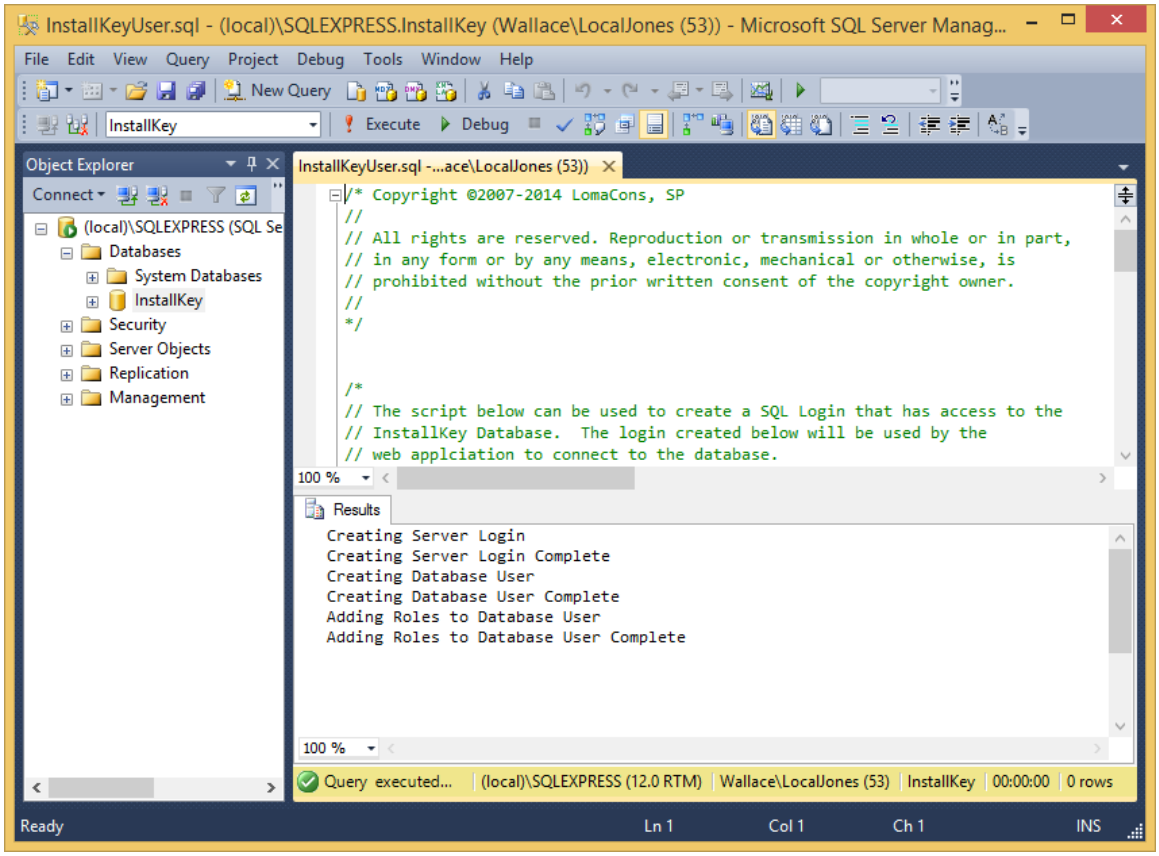
Database Creation

2. Unzip the Database.zip file
3. Execute the InstallKeyDatabase.sql script file on the InstallKey database. This will create the database schema and load the initial fixed content. As this script executes, you should see each step in the process complete in the results pane. Confirm that the script ran successfully and without error by reviewing the contents of the results pane.



Database Schema Creation

4. Execute the InstallKeyUser.sql script on the InstallKey database and confirm that the script ran successfully and without error by reviewing the contents of the results pane. This will create a SQL login with a password and associate the login with the InstallKey database. This login will be used to connect to the database from the web application. As a security best practice, this login is configured with minimal access constraints such that it has access to only the InstallKey database with datareader, datawriter and execute permissions.

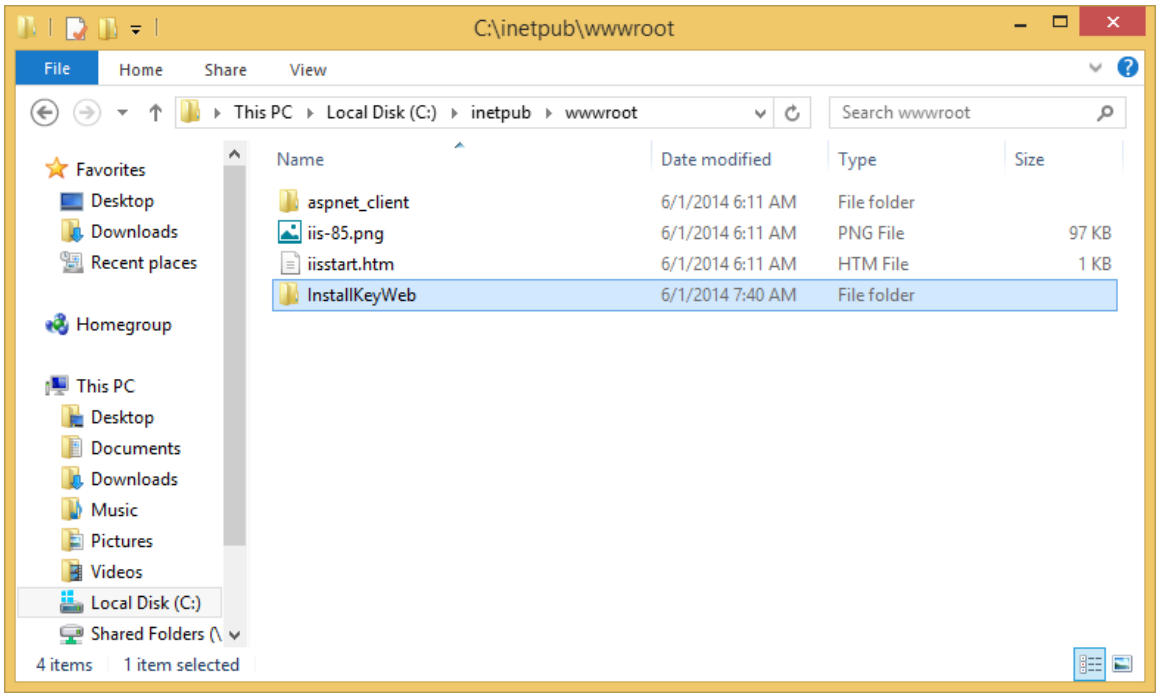


Database Login Creation

Web Application Creation and Setup

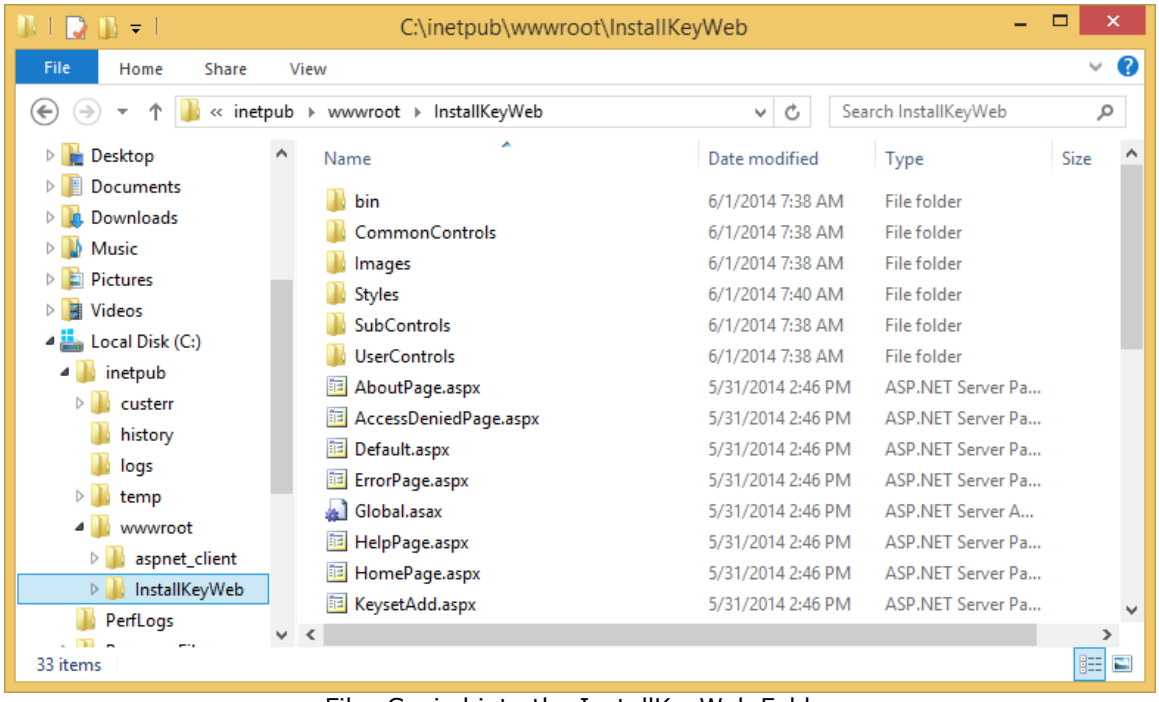
To create the InstallKey web application, follow these steps. These steps assume a default installation of IIS with the Default Web Site and a default installation of the .Net Framework 4.0 with the ASP.Net v.40 application pool.

1. Create the InstallKeyWeb folder on the file system in the C:\inetpub\wwwroot folder.



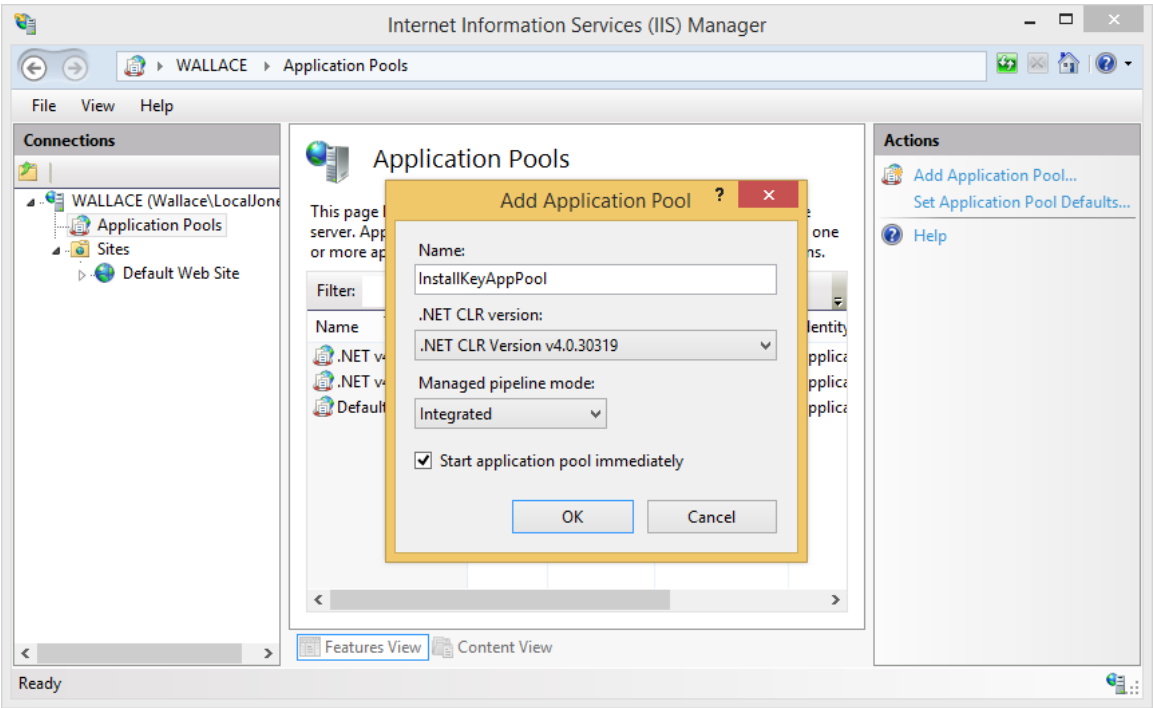
Create the InstallKeyWeb folder

2. Unzip the InstallKeyWeb.zip file.
3. Copy the files and folders from the zip file into C:\inetpub\wwwroot\InstallKeyWeb



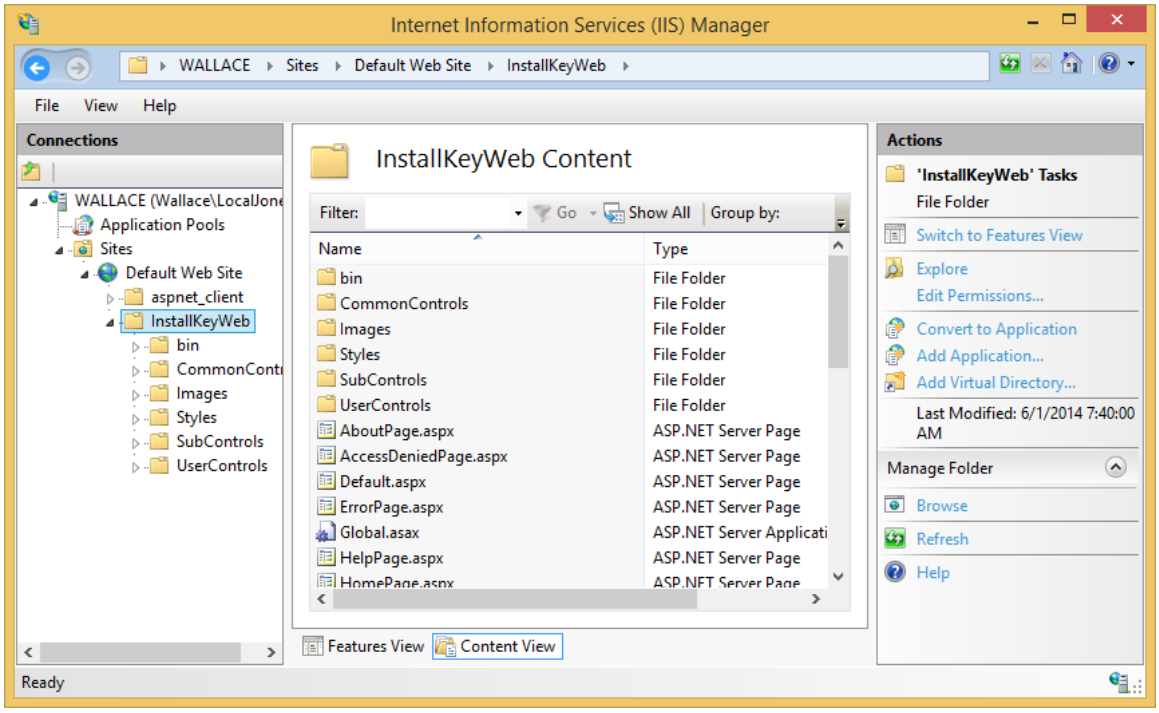
Files Copied into the InstallKeyWeb Folder

4. Open IIS Manager and select the Application Pools node in the tree. Choose the option to Add Application Pool. Name the application pool InstallKeyAppPool and configure the application pool with .Net Version 4 and pipeline mode set to Integrated.

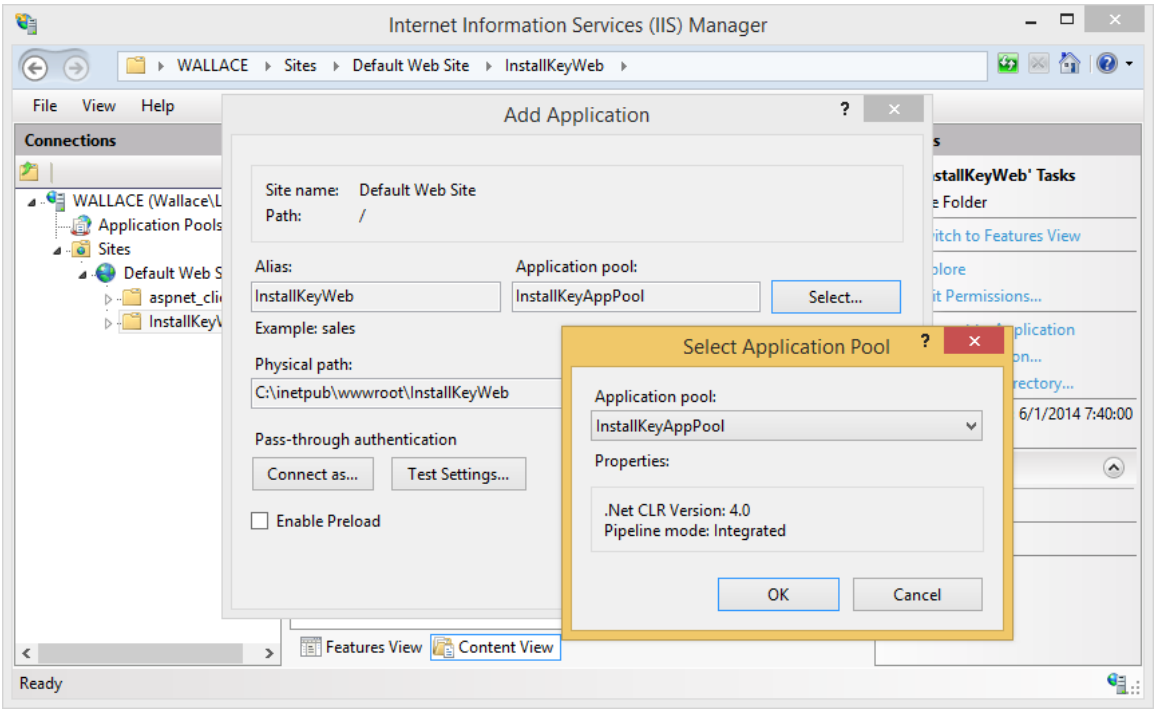


Creation of the InstallKey Application Pool

5. Expand the Default Web Site node in the tree. Select the InstallKeyWeb folder. Select Content View. You should see the same files that were copied to the file system.

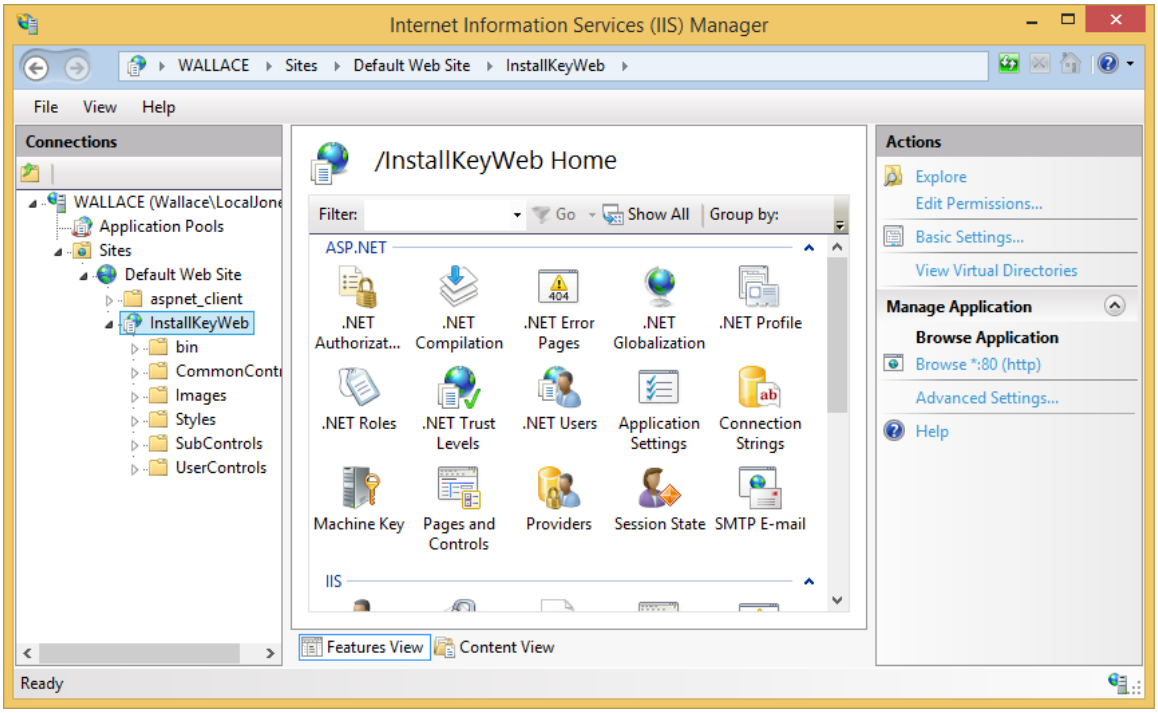


6. Right Click on the InstallKeyWeb folder and choose the option to Convert to Application. The Alias and Physical Path Settings should already be correct. Click the Select button to change the Application Pool. Choose the InstallKeyAppPool. (This app pool must be .Net 4 and Integrated pipeline.)



Convert to Application Dialog and Select Application Pool

7. Click Ok to select the Application Pool, and click Ok to create the web application in IIS.



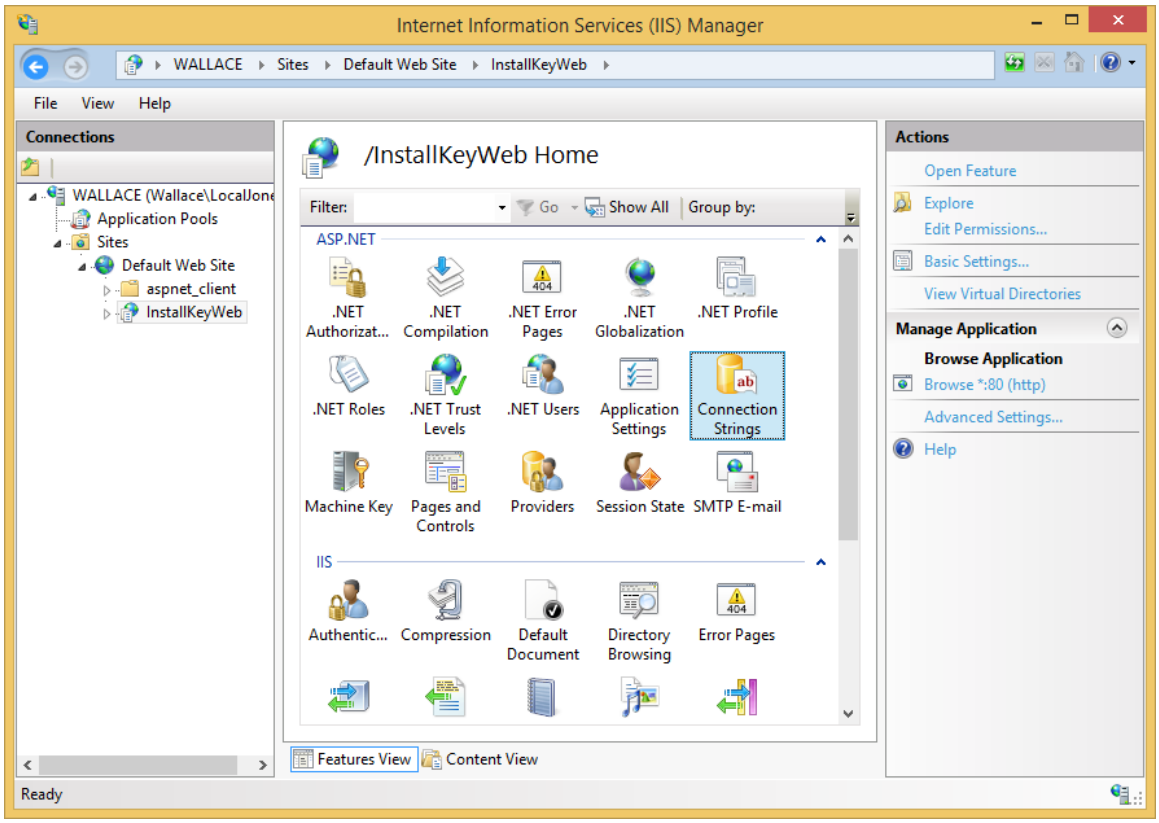
Completed IIS Application Configuration

Database Connection String

The database connection string in the web.config file may need some changes that are specific to your environment. The connection string can be set either by modifying the web.config file directly, or by editing the connection string from within IIS.

Edit the Connection String From Within IIS

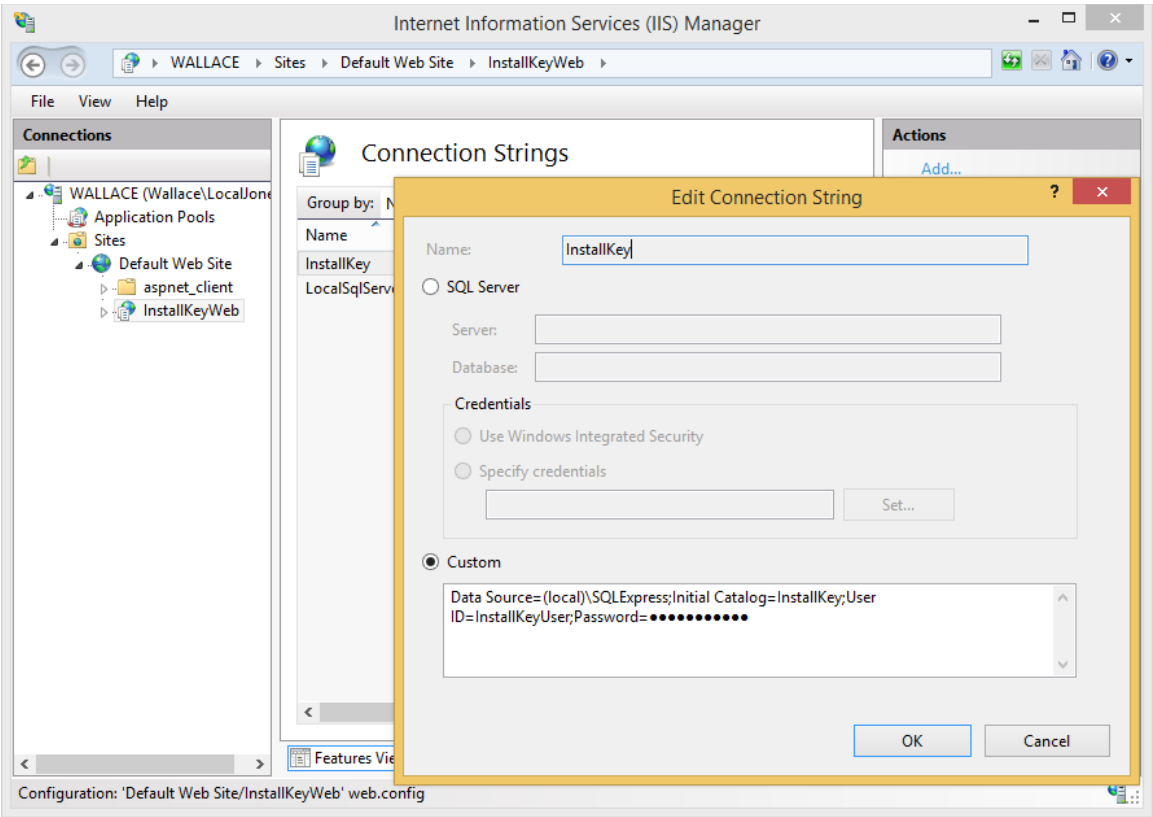
The easiest way to set the connection string is to edit it from within IIS Manager. To edit the connection string from within IIS, navigate to the InstallKeyWeb application, and open the Connection Strings settings.



Location of Connection String inside the Web Application in IIS

If you followed the steps above using the provided database name and SQL login creation scripts, then the provided connection string values for Initial Catalog, User ID and Password values should be correct, but you will need to confirm and configure the Data Source for your environment.

Confirm that the Data Source is correct for your running instance of SQL Server. If you are using SQL Express on the same machine as the web server, the Data Source will most likely need to be changed to (local)\SQLEXPRESS



A Database Connection String (yours may be different)

Directly Editing the web.config File

Alternatively, the connection string can be set by editing the web.config file. If you are comfortable with directly editing configuration files, you can set the database connection string directly instead of using IIS Manager.

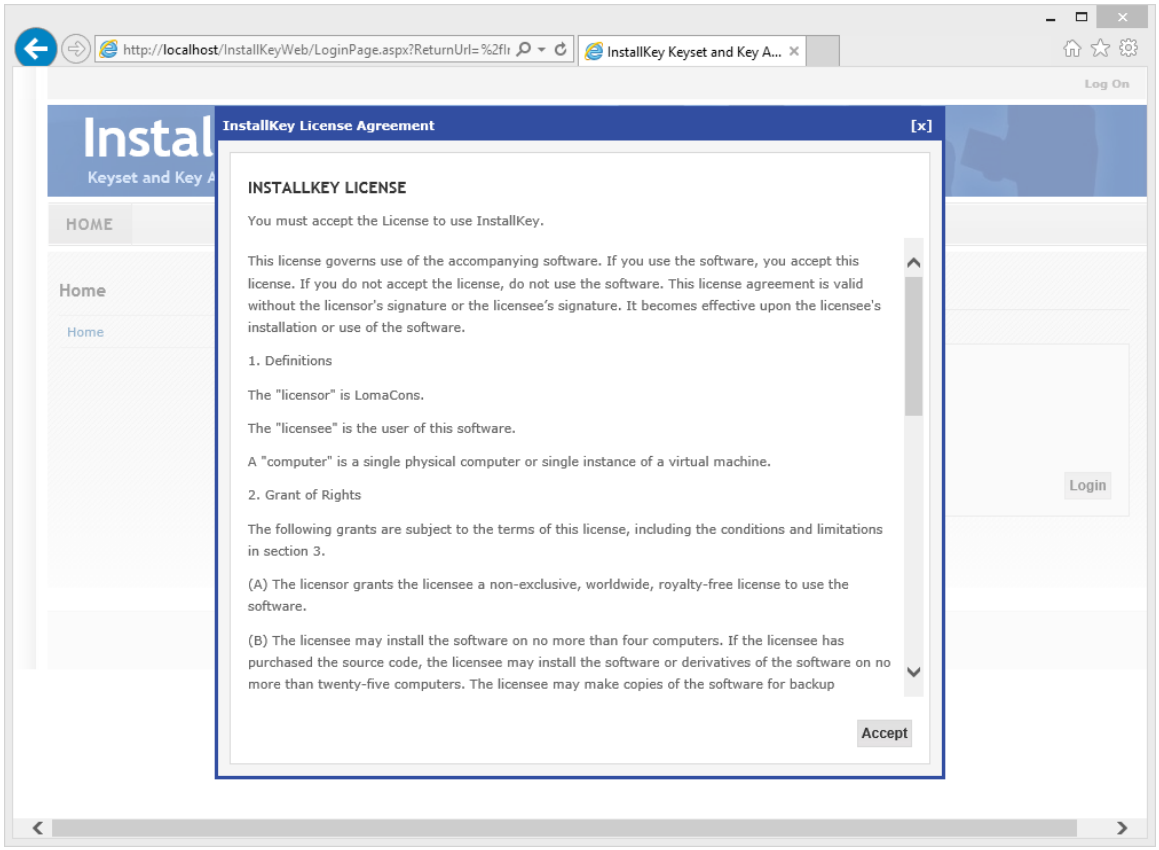
Depending upon your machine configuration, Windows UAC (User Account Control) may not allow you to directly edit and save changes to the web.config file, even when you are logged in as an administrator. To resolve this, copy the web.config file to your My Documents folder, edit the web.config file and copy it back, overwriting the original web.config file. (Alternatively you can disable UAC entirely by changing the User Account Control: Turn on Admin Approval Mode setting using the Local Policy editor, however this decreases overall operating system security.)

Confirming the Installation

To confirm the installation is complete and functioning properly, open Internet Explorer and navigate to the InstallKey Application URL. If your browser is on the same machine as IIS, the url below should work.

http://localhost/InstallKeyWeb/Default.aspx

If the database, web application and connection string was setup properly, you should be navigated to the Login Page and be presented with the License Agreement. Please review the agreement and click Accept.



Install Key License Agreement

Common Issues and Solutions

The most common installation issues are listed below. Additional installation guidelines and tips can be found on the FAQs on the www.lomacons.com website.

Cannot Edit the web.config File or other Files under C:\inetpub

Often it is not possible to edit files directly under the C:\inetpub\ folder, even when logged in as an Administrator. This is a security feature of Windows UAC (User Access Control.) There are two methods to resolve this.

1. Copy the web.config to your My Documents folder or any other folder where UAC allows file editing. Edit the file, save the changes. Copy the file back into folder under C:\inetpub by overwriting and replacing the original file. This is the most secure method as it does not violate Microsoft intended security policies.
2. Disable UAC in the local security policy. To disable UAC, open the security policy editor, and find the setting for User Account Control: Turn on Admin Approval Mode. Set this property to disabled and restart the machine. Note, that this method is less secure as this disables UAC. If you choose to use this method, it is recommended that you re-enable this setting after making your edits to the web.config file.

HTTP Error 500.21 – Internal Server Error

You may see HTTP Error 500.21 – Internal Server Error the first time you open the http://localhost/InstallKeyWeb url in a browser after installing. This error usually indicates that support for ASP.Net 4.0 is not installed in IIS properly.

Follow these steps to resolve

1. Confirm that IIS and support for ASP.Net is enabled. Open the Turn Windows Features on or off dialog. This dialog is located in the Add/Remove Programs applet in Control Panel.
2. If IIS and ASP.Net is enabled, but you still see the error, you may need to re-register support for ASP.Net 4.0 into IIS. Open a command prompt as an administrator and run the command below %windir%\Microsoft.NET\Framework64\v4.0.30319\aspnet_regiis.exe -i

Web App Does Not Display in the Browser

Even after following and reviewing all the installation steps above, the login web page does not appear when attempting to connect to the web application.

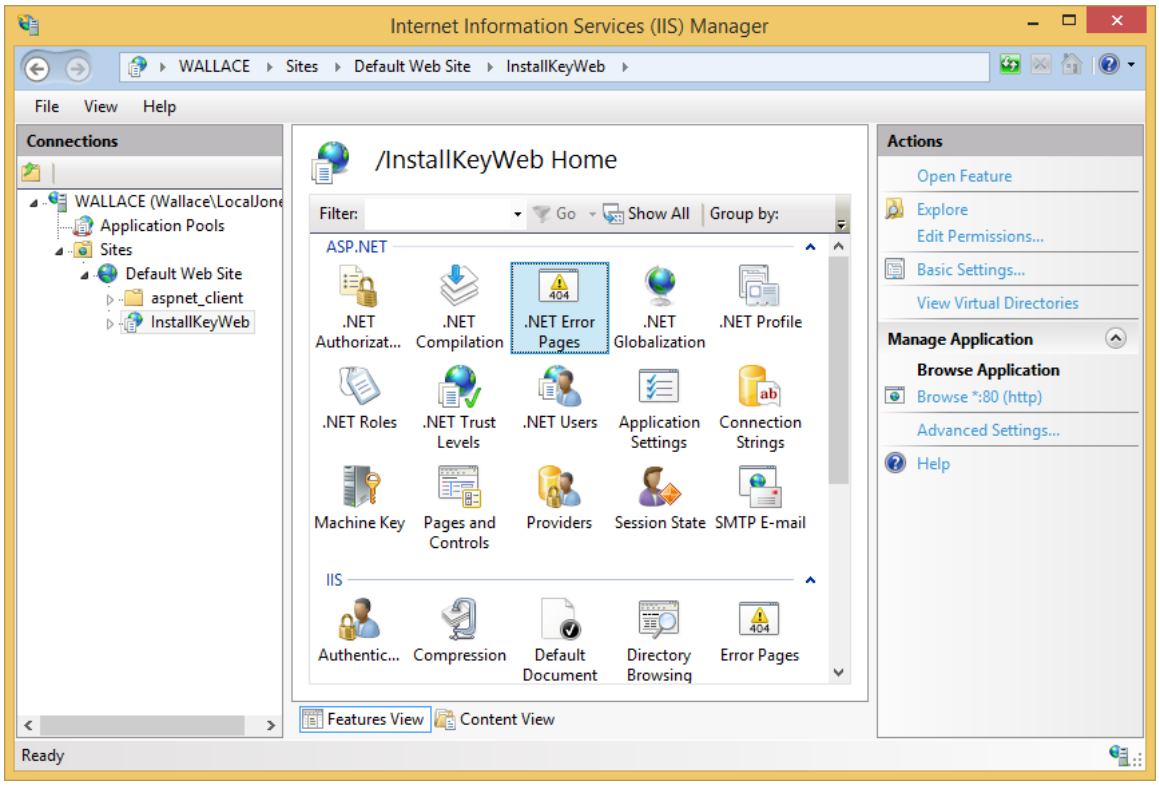
The steps above assume that you are installing the InstallKey web application on a fresh install of Microsoft Windows, and should work in most all installations. However, it is possible that your environment or network administrator has configured the operating system differently such that the web application does not work properly. When there is an error, the InstallKey web application should display detailed error information and exception details which may help in locating the source of the error.

However, there are times where something may be preventing the InstallKey error page from displaying properly. If this occurs, you will need to enable the standard Microsoft ASP.Net error web page to see the error. Change the customErrors mode to Off and restart the web site. The customErrors is configured in the web application's web.config file or it can be set from within IIS.

To set the custom errors off in the web.config file, look for a line similar to the one below and change the mode value to Off. Once you have resolved the problem, you can change the mode back to On.

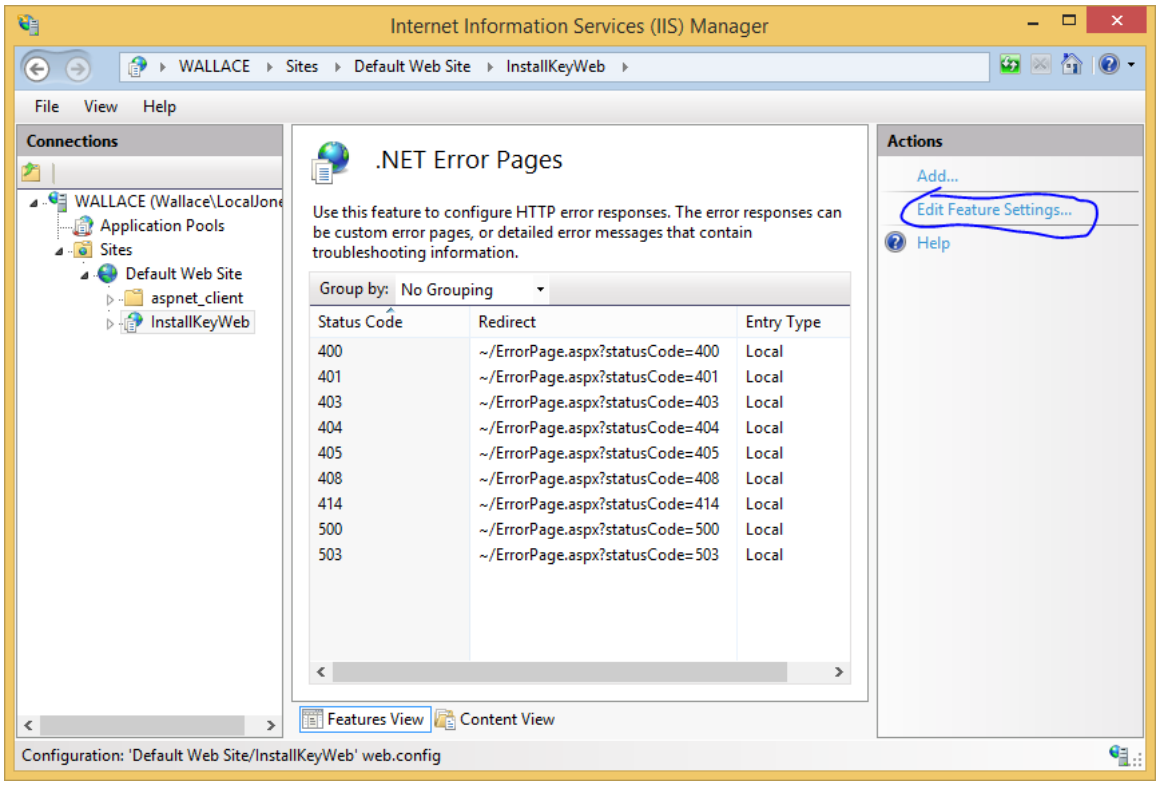
```
…
<customErrors defaultRedirect="~/ErrorPage.aspx" mode="Off">
…
```

To set the custom errors to off from within IIS, open IIS, navigate to the InstallKeyWeb application, and open the .Net Error Pages settings. (This method will make the same change as editing the web.config.)



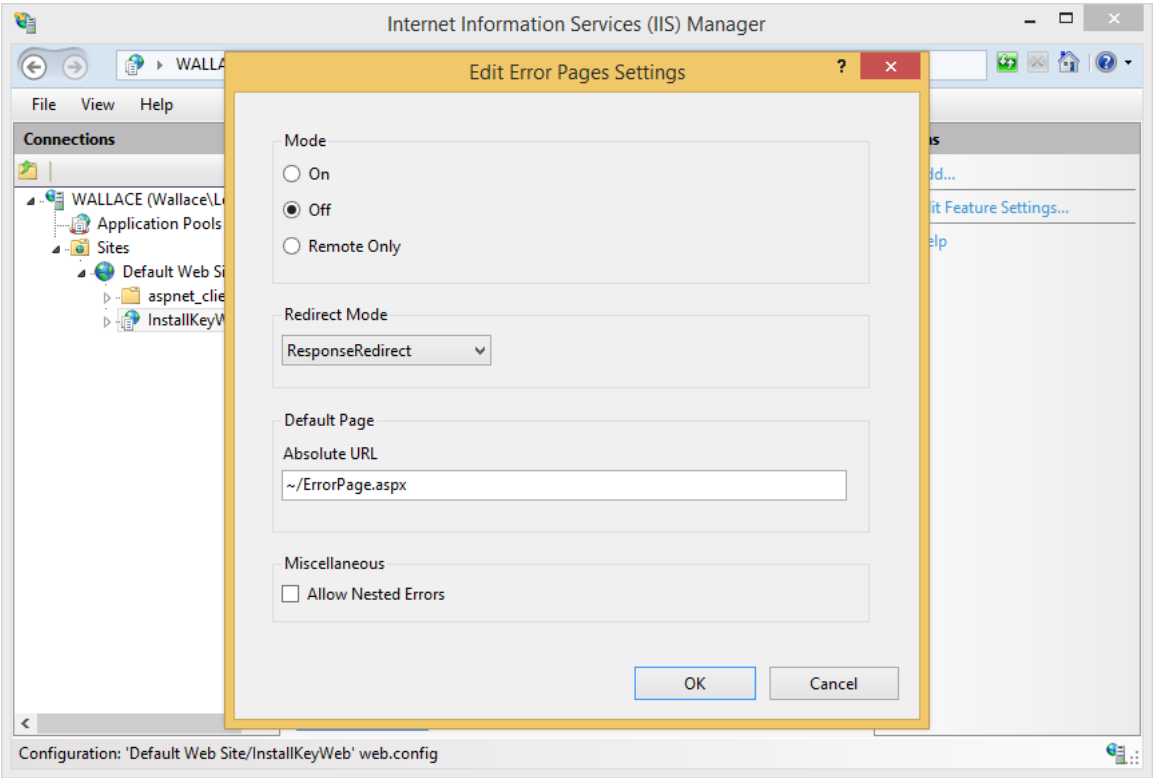
Location of .Net Error Pages Settings

In the .Net Error Pages click the link for Edit Feature Settings



Location of Edit Feature Settings…

In the Edit Error Pages Settings, set the Mode to Off and click OK.



Edit Error Pages Settings Dialog

Once you have resolved the problem, you can change the mode back to On. This will enable the normal InstallKey error processing.

Upgrading from Prior Releases of InstallKey version 5.x

Upgrading the web application from a prior release of version 5.x requires that you apply any database changes, and replace the web application and web.config files.

As with any upgrade, you should test the upgrade before making any changes to your production environment. At a minimum, you should create a backup of your database and web.config file.

Follow these steps to upgrade your InstallKey server. These steps assume that you have followed the steps above to install InstallKey and that your web application files were copied into C:\inetpub\wwwroot\InstallKeyWeb.

1. Unzip the Database.zip file.
2. Execute the InstallKeyDatabase.sql script file on the database. This will apply schema changes and add any additional fixed content to bring your database up to the current release. This script will only apply the necessary changes, and will not delete any of your keysets or keys.
3. Delete all the files under the C:\inetpub\wwwroot\InstallKeyWeb including the web.config file.
   - Do not delete the C:\inetpub\wwwroot\InstallKeyWeb folder.
   - Take note of the database connection string before deleting the previous web.config file. This will need to be recreated later in the new web.config file.
   - It is important that all files and folders are deleted so that all files and folders from the prior release are removed.
4. Unzip the InstallKeyWeb.zip file.

5. Copy the files and folders from the zip file into C:\inetpub\wwwroot\InstallKeyWeb
6. The database connection string will likely need some changes that are specific to your environment. Open the web.config file; edit the connection string to connect to the InstallKey database. The database connection string will most likely be the same as was in the old web.config file.
7. If you have any custom pre-validate or post-validate code, these will need to be updated. To update them, you will need to recompile your custom code referencing the newest LomaCons.InstallKey.Common.dll assembly and re-install your new dll assembly.
8. If you are exposing the WCF web service endpoints over https, you will need to ensure that you have a valid SSL certificate properly installed, and the https binding configured. Then uncomment the secure endpoints in the web.config.

Support for HTTPS (HyperText Transfer Protocol Secured)

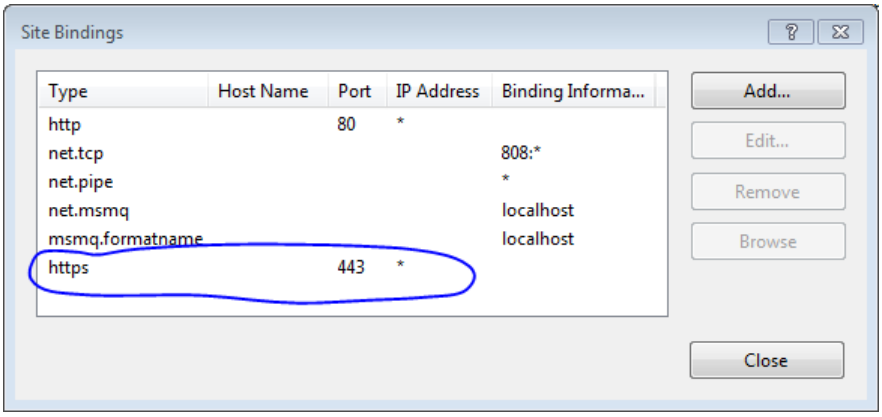
InstallKey is designed such that the validation web service will be tamperproof when running under the http protocol. Although the data cannot be modified and is safe from tampering, the content sent to and from the web service can be seen if someone is viewing the raw content as it travels across a network. (This is intentional and is just the very nature of how HTTP works.) If your connection to the web application and the API web services is internal, on the same machine, or within an intranet environment, then transmitting the data in the clear is not likely to be an issue.

However, if you are exposing the web application or the API web services over the internet, you may want to run InstallKey over HTTPS. This will enable this industry standard security protocol for encrypting the raw web requests and responses. To enable InstallKey to run over https, your will need to configure your web site with a valid SSL certificate and adjust the web.config file for the WCF endpoints accordingly.

Because InstallKey uses standard Microsoft tools and technologies, there is nothing unusual about enabling HTTPS. If you are knowledgeable about configuring IIS and HTTPS, you will have no troubles making the necessary changes to your installation of InstallKey. To configure InstallKey for HTTPS, you will need to have an IIS server with a valid SSL certificate and uncomment the secure endpoints defined in the InstallKey web.config file.

Follow these steps and guidelines for enabling InstallKey to work over HTTPS.

1. Install a valid and functioning SSL certificate into your IIS web server from a trusted certificate authority. Contact your certificate authority for details about how to purchase, install and set the binding for the certificate into IIS.
2. In IIS Manager, view the Bindings for the Web Site. Confirm that the IIS web site has the https binding installed. At this point you should be able to view the InstallKey web application in a browser over HTTPS.



Web Site Binding Configuration in IIS Manager

3. To enable the WCF web services to function over HTTPS, you will need to expose the secure endpoints defined in the InstallKey web.config file. In the web.config, locate and uncomment the two lines below. This will allow IIS to load and activate the secure WCF endpoints.

Caution – DO NOT uncomment the two secure endpoint definitions without first adding the https binding to IIS. If the web site does not have a valid https binding and the lines are uncommented, then all WCF services will fail to load, including the http service.

```
...
<endpoint binding="basicHttpBinding" bindingConfiguration="secureHttpBinding"
                            contract="LomaCons.InstallKey.Web.Services.IValidator"/>
...
<endpoint binding="basicHttpBinding" bindingConfiguration="secureHttpBinding"
                            contract="LomaCons.InstallKey.Web.Services.IApiProcessor"/>
...
```

Secure HTTPS Endpoints

4. No changes are necessary in your calling application. To configure the calling endpoint in your application, call the SetServiceAddress() method passing in the URL to the validation service. The SetServiceAddress method will automatically detect if the URL scheme is http or https and adjust the client endpoint binding accordingly.

```
string validationUrl = @"https://.../InstallKeyWeb/Validator.svc";
ClientSideValidator validator;

// create an instance of the ClientSideValidator object
validator = new ClientSideValidator( keysetName,  password,  factor,  validationCode);
// set the service address URL
validator.SetServiceAddress(validationUrl);
```

Client Side Setup for Calling the Validation Service over HTTPS

6. Key Management Website

The LomaCons.InstallKey.Web website provides a simple web based user interface to manage your keysets and validation keys. Once you have logged in to the application, you can create keysets and keys. If you have purchased a license for InstallKey, you will be able to create an unlimited number of keysets and keys.

If you do not yet have a license, InstallKey will function as an evaluation, and will limit you to creating one keyset and three keys. All other functionality is available, and will allow you to fully evaluate the product before you purchase a license.

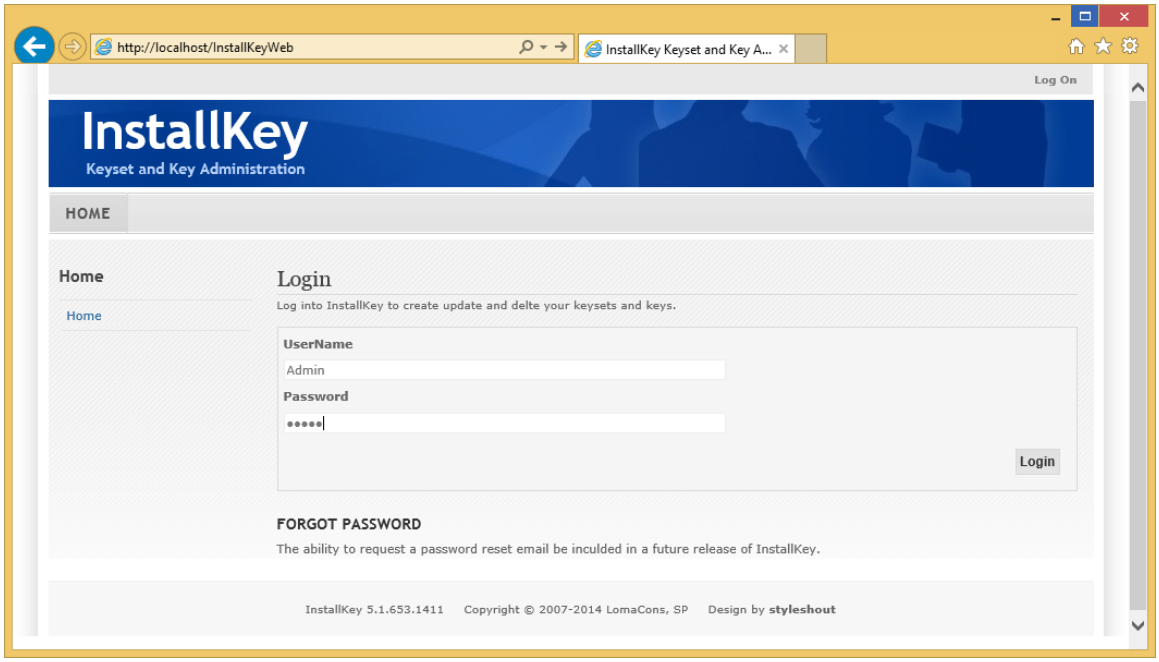
Login

Access to the web application is protected with ASP.Net Forms Authentication. This type of authentication validates username/password combinations against the values stored in the InstallKey database.

To login for the first time, open a browser window and navigate to the InstallKey web application. If you installed the application locally, as described in the preceding chapters, you should be able to connect to the website at this URL

```
http://localhost/InstallKeyWeb/Default.aspx
```

If everything is configured properly and you have entered the correct URL, you should see the login page below.



Login Page

The default username and password for a new installation was created when the database was created. To log in for the first time, use the Admin login and password below.

```
UserName:  Admin
Password:  admin
```

It is strongly recommended that you change the admin password using the user management or user preference settings (see below.)

Login Maintenance

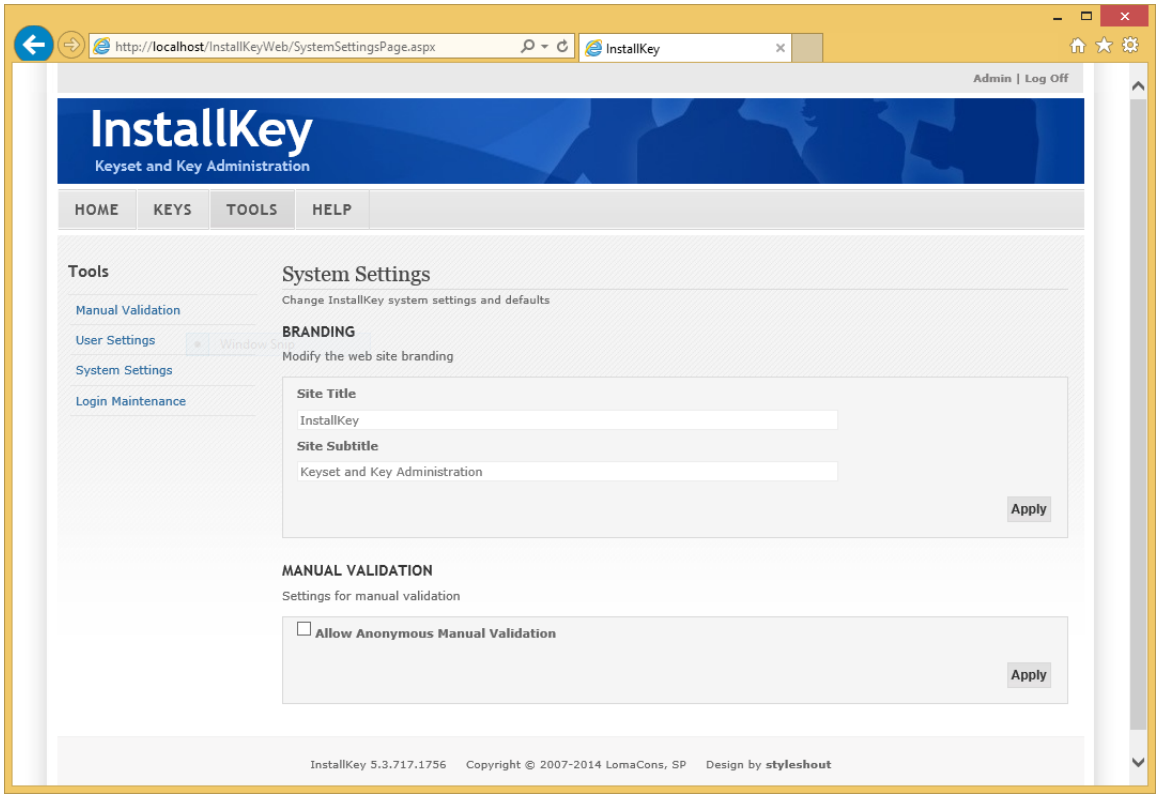
If you have multiple users who will be managing you keysets and keys, use the login Maintenance pages to create the logins, and reset passwords. Additional user access rights may be set to allow users to access the System Settings, Keysets, Sure Keys and the API.

User Settings

Users can change their login password at any time from the User Settings page. Additional user editable preferences will be added as needed in future versions of InstallKey.

System Settings

There are a few system settings that can be set to configure how InstallKey functions. These settings can be modified at any time from the System Settings page.



System Settings Page

Branding

The branding settings allow you to modify the Title and Subtitle displayed on the InstallKey web application. This may be useful if you want to display something specific to your business, instead of the normal InstallKey title.

Manual Validation

This section allows you to modify access to the manual validation web page. By default, you must first log into the InstallKey Web application with a username and password before validating a surety file.

When the Allow Anonymous Manual Validation setting is enabled, anyone can access the manual validation page (without logging in first.) This may be useful if you want to allow your customers without internet connections to self-validate surety files.

Pre Validate and Post Validate Custom Code Integration

These two sections allow you to define a custom .Net assembly that will be called during the pre or post sure key validation process. See the chapter below on Customizing the Validation Process for more information about how to create and integrate your custom code.

API Settings

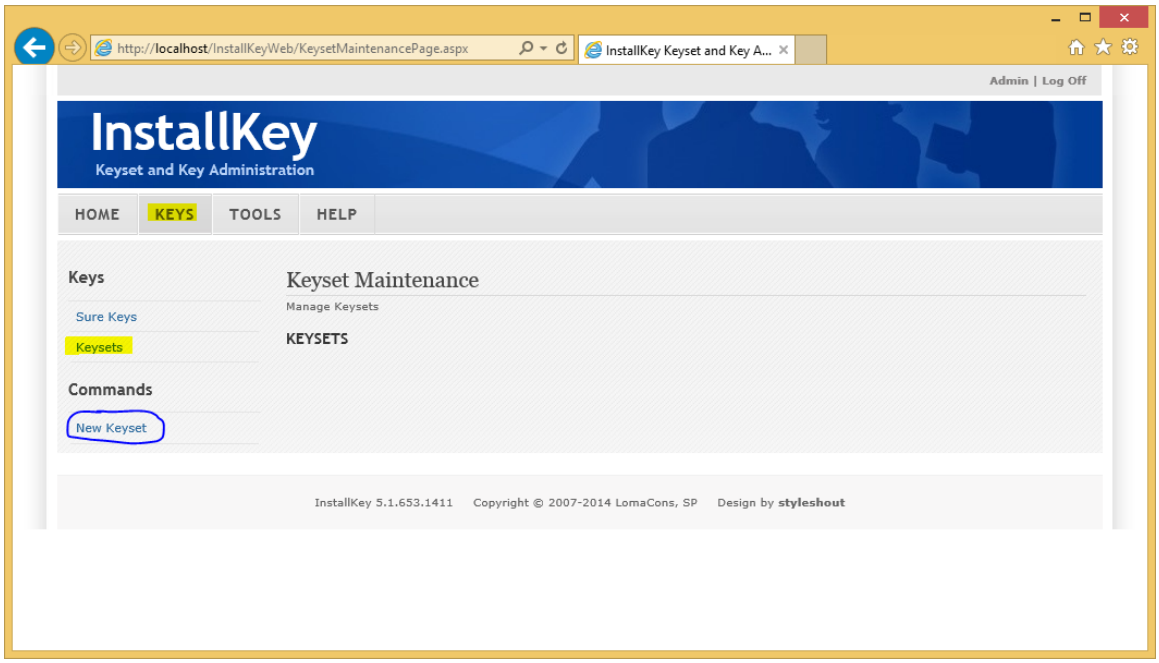
This section contains the settings and values that will be used with the InstallKey API. See the chapter below on the InstallKey API for more information about how to utilize the API.

Keysets

A Keyset is a logical grouping of keys. Typically, a separate keyset should be created for each unique software product you want to support key based validation.

The keyset is identified by a name, and consists of a password and a keyset factor. The keyset factor is used to programmatically identify a humanly readable key as being of that keyset type. The password is used to perform encryption and key information validation. This encryption and validation is used for preventing casual tampering of an InstallKey surety file and it is used during server validation of the key.

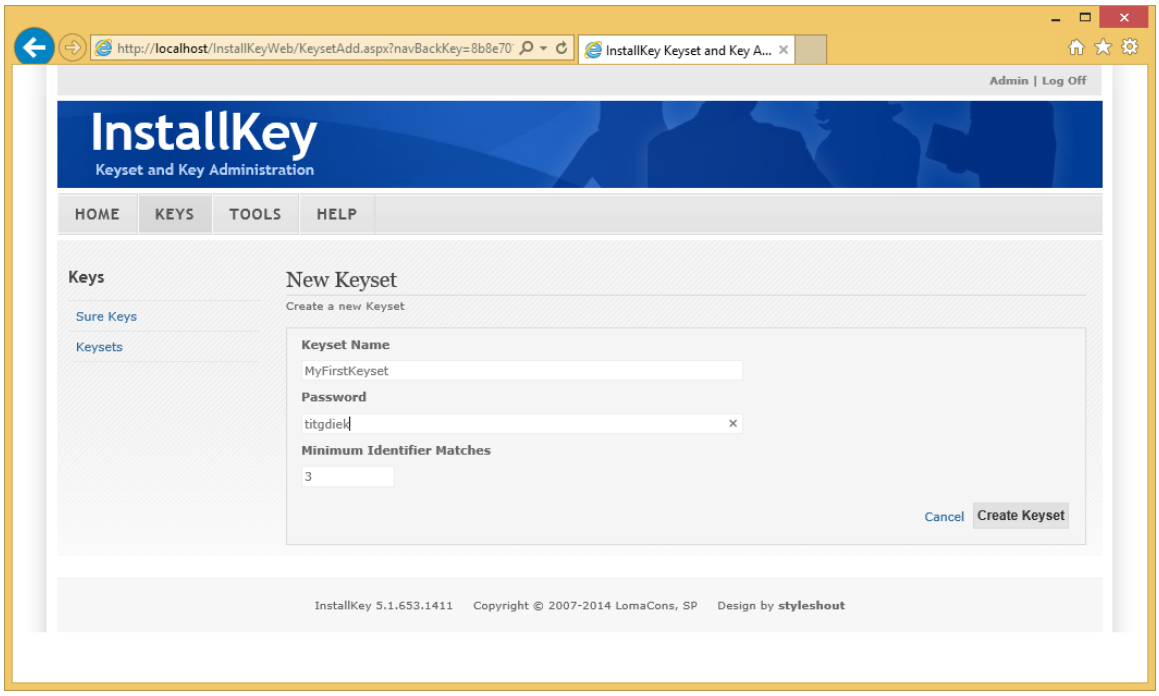
Keysets are managed from the Keys tab. To begin, navigate to the Keys tab and select the Keysets option from the Keys sidebar. If you have already created a keyset it will appear in the list displayed. However, if this is your first keyset, the list will be empty as pictured below.



Keyset Maintenance Page

Creating Keysets

To create a keyset, click the New Keyset link on the Commands menu. This will navigate you to the New Keyset page.



Create a new Keyset

Enter a keyset name and a password. Set the Minimum Identifier Matches to 3. Three is the default value to use when using the default machine identifier matching provided by InstallKey.

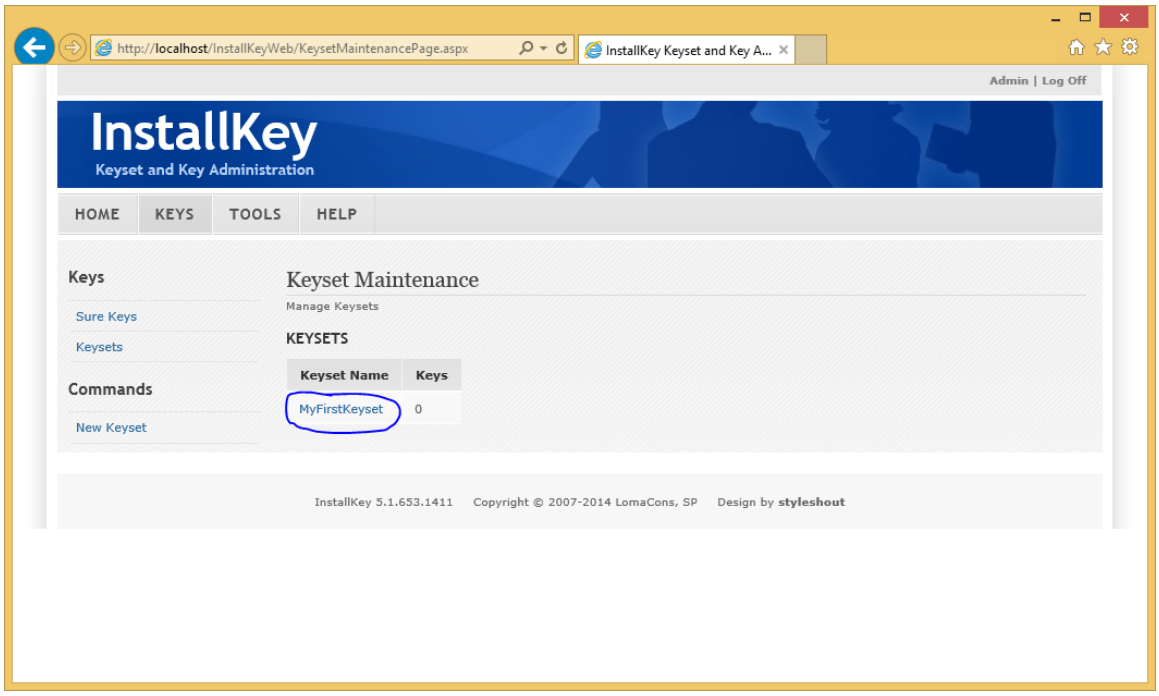
Note: The minimum identifier matches value may need to be different if you are using custom client identifier values. See the section below titled "Custom Identifiers and Matching" for more information.

Click the Create Keyset button to create the keyset. This will automatically generate a keyset factor that matches this keyset name and password.

Note: Once you have created a keyset, its password cannot be modified. Although there are no password restrictions, it is recommended that you use a password that is sufficiently complex.

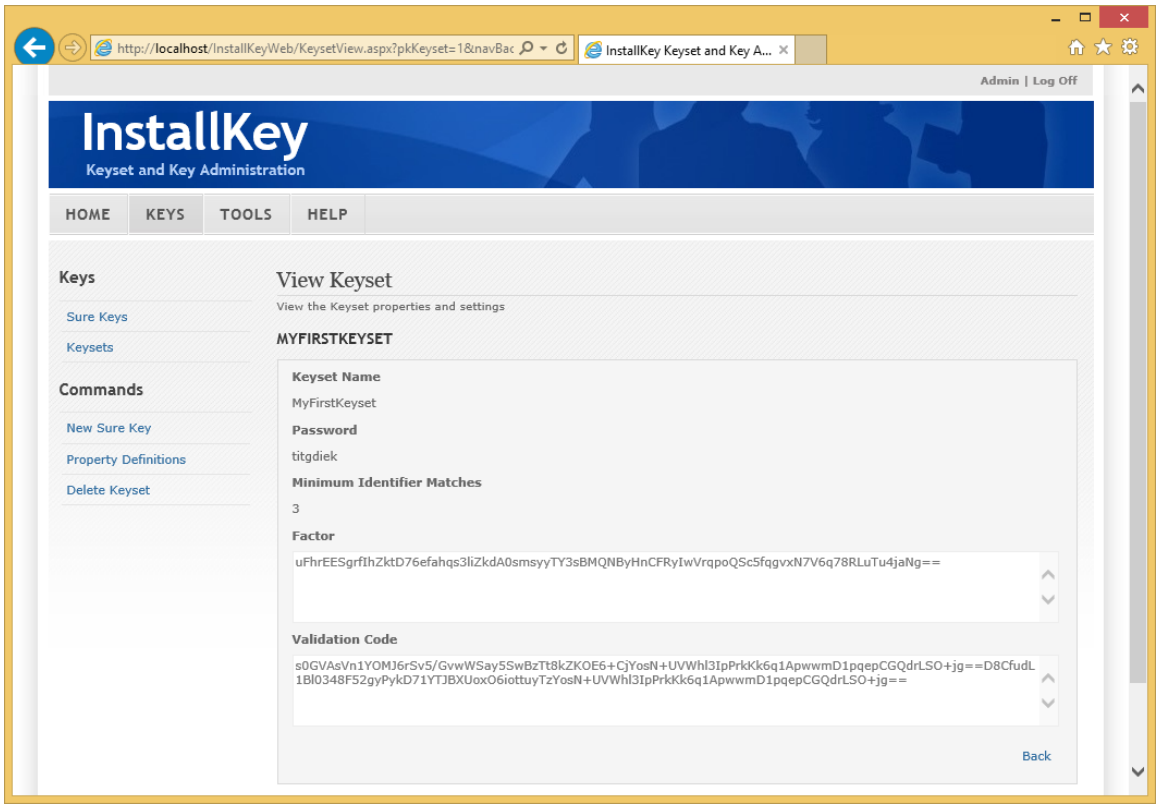
Note: No two keysets are the same, even when the keyset name is the same. For example, if a keyset is deleted and a new keyset is created with the same keyset name, the keyset factors will be different.

To view the newly created keyset, click on the keyset name in the list



Keyset List

The keyset details page will show the keyset name, password, factor and validation code. These values will be used later to validate keys on the client and as the parameters to the ClientSideValidator object.



Keyset Details

The validation code is a uniquely generated value. It will be different every time the keyset details are viewed.

Editing Keysets

Keysets cannot be modified. Once created, the keyset name, password and factor are fixed values.

Deleting Keysets

Keysets can be deleted, click the Delete Keyset command to delete the keyset that you are currently viewing. A confirmation popup will appear to confirm this action.

If the keyset has one or more active keys, the keyset cannot be deleted. To delete such a keyset, first revoke all of the sure keys in the keyset, and then delete the keyset.

Deleting the keyset will also delete all related sure keys and properties, and cannot be recreated.

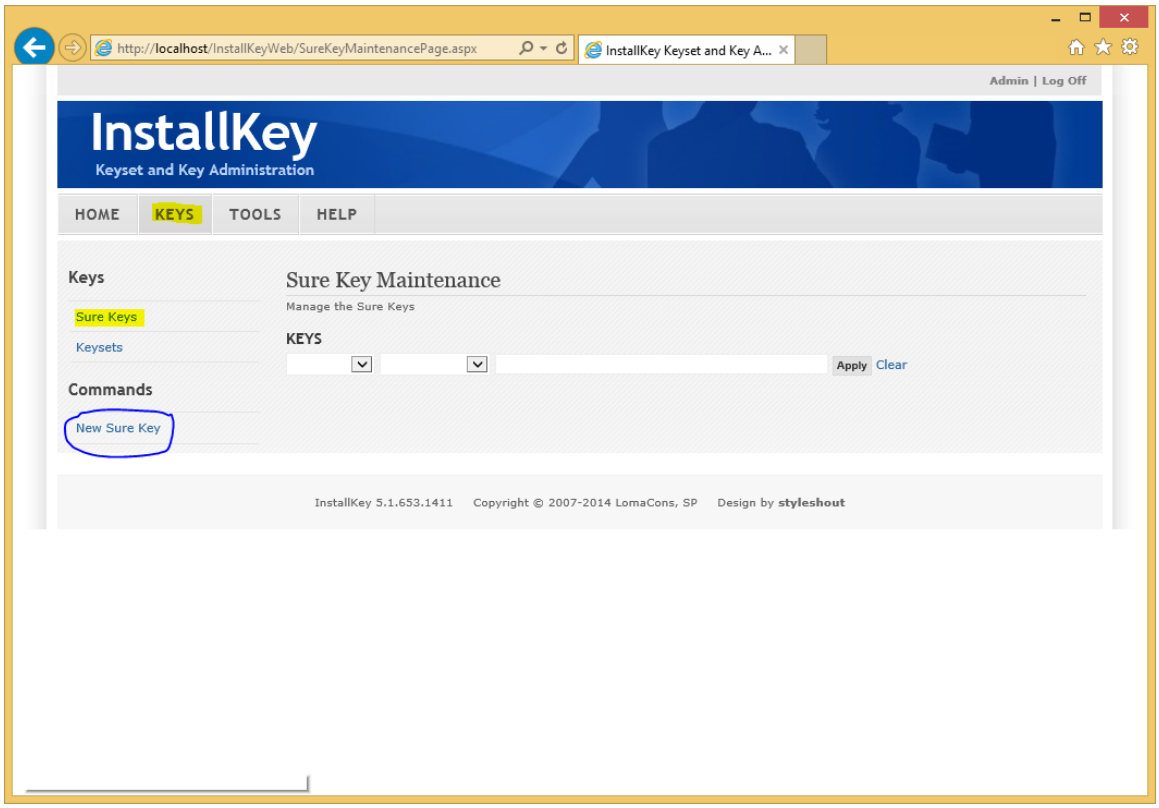
Note: No two keysets are the same, even when the keyset name and password are the same. For example, if a keyset is deleted and a new keyset is created with the same keyset name and password, the keyset factor will be different.

Sure Keys

After you have created a keyset, you can create keys for that keyset.

Keys are managed from the Keys tab. To begin, navigate to the Keys tab and select the Sure Keys option from the Keys sidebar. If you have already created a key it will appear in the list displayed. However, if this is your first keyset, the list will be empty as pictured below.

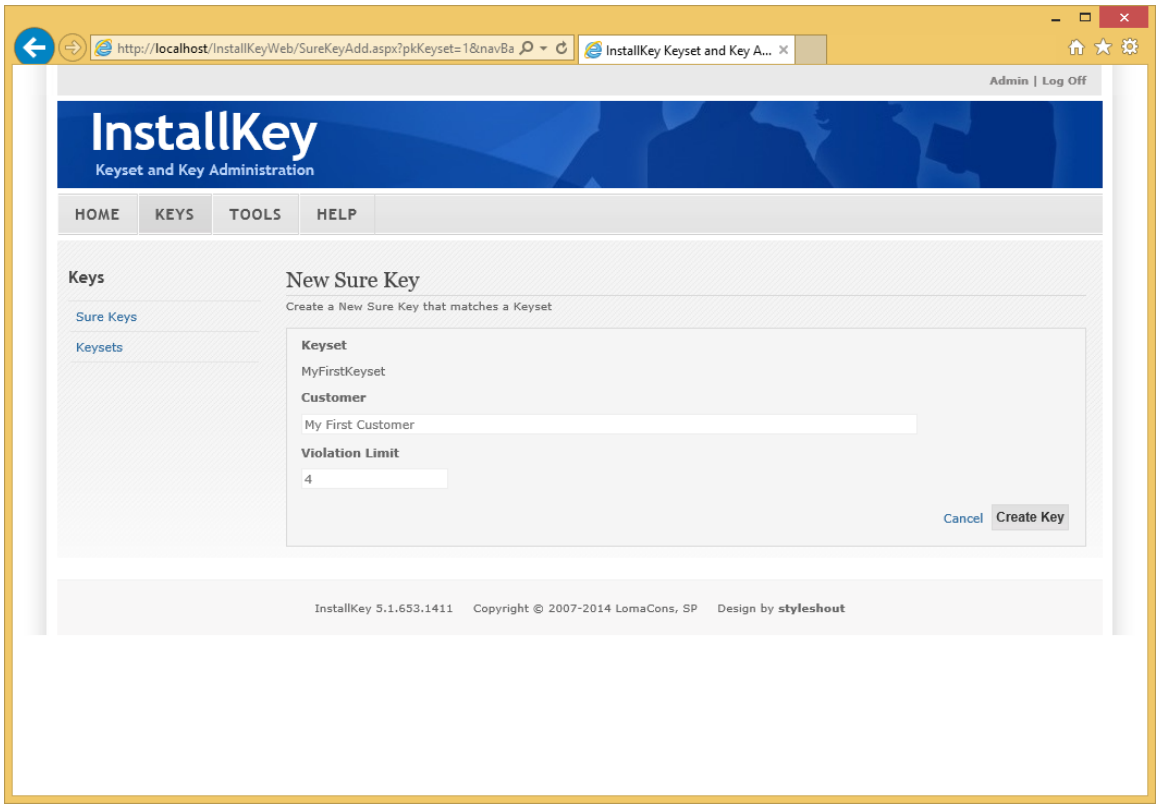
Each 30 character key is unique and is a match to its keyset.



Sure Key Maintenance Page

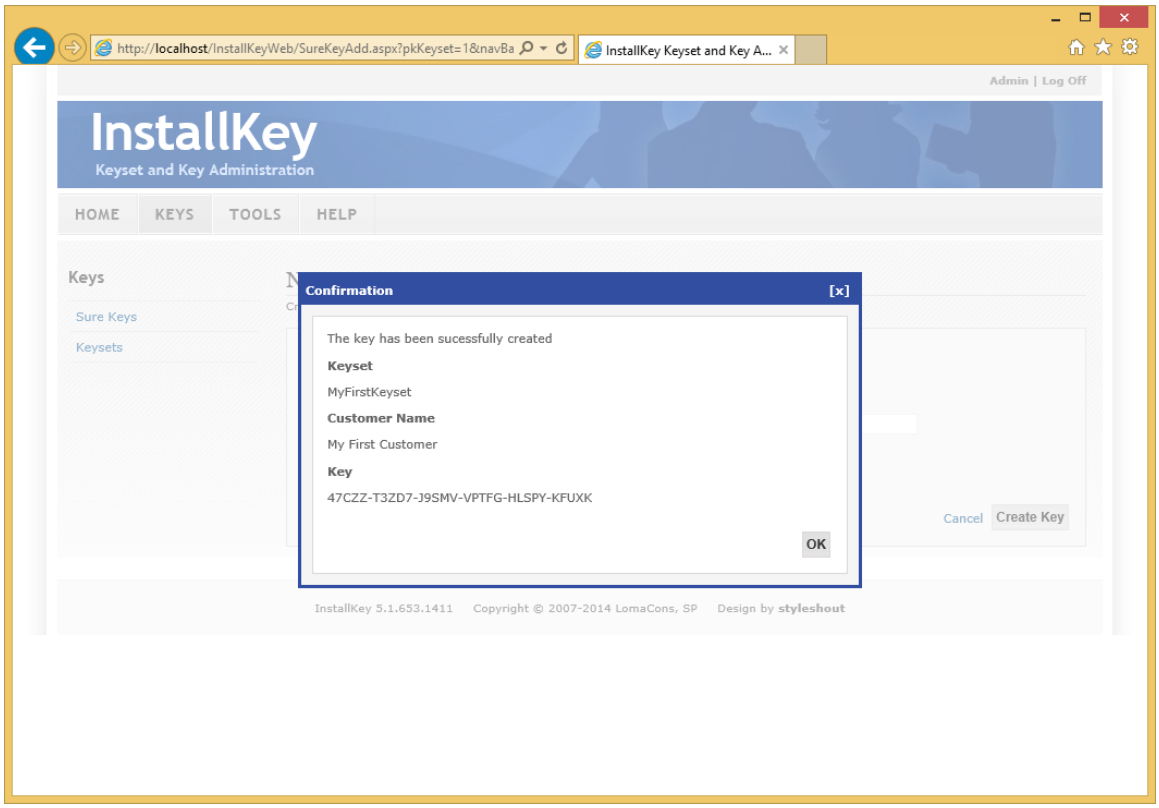
Creating Keys

To create a key, click the New Sure Key link in the Commands menu. This will navigate you to the New Sure Key page.



Create a New Sure Key

Enter the Customer Name and a Violation limit. Click the Create Key button to create the key.



Key Creation Confirmation

The violation limit is the number of times that a key can be uniquely validated. The client specific, unique identifiers (such as the bios identifier, hard disk, CPU ID, etc. or other values) are used to uniquely identify where the key was validated. The usage count is incremented only when a new set of identifiers is identified. Thus repeat validations from the same machine (with the same identifiers) will not increase the usage count.

For example, if the key violation limit is 4 and your customer attempts to validate his key on 5 machines, the first 4 will succeed, but validating from the fifth machine will fail. Even after the fifth machine fails, only the first four machines will continue to validate successfully. Your customer will have to contact you to increase the violation limit, thus keeping you in control of number of valid installations.

Editing Keys

Use the grid filter and sorting to find the key. To edit a key, select the key from the list. Click the key to navigate to the key edit page. On the Key Edit page, you can change the customer name and violation limit.

Future releases of InstallKey will provide additional usage counting and validation information.

Revoking Keys

Revoking a key effectively disables the key at the server. Any future attempts to validate the key will fail regardless of how many times the key has been used.

Deleting Keys

Use the grid filter and sorting to find the key. To delete a key, select the key from the list to navigate to the key edit page. On the Key Edit page, you can delete the key by clicking the Delete Key command. A delete confirmation dialog will appear.

To prevent accidental deletion of a key, the key must first be revoked. Revoke the key by selecting the Revoke checkbox, and Save the changes. Then click the Delete Key command to delete the key.

Note that deleting a sure key is permanent and the key cannot be recreated because this will delete the key, the usage history and usage count records from the database. All history and information about the key will be deleted. If you are unsure about deleting a key, it may be better to revoke it instead.

7. Client Side Tester and Other Samples

InstallKey includes a number of client side applications and code samples written in C# and VB.Net. These samples provide working code that utilizes the ClientSideValidator object to confirm that a key was successfully validated.

The InstallKey client side libraries are 100% compatible with VB.Net. Samples written in VB.Net will be included in a future release of InstallKey.

Client Side Tester

The client side tester is a single win form application that can be used to test and validate a key against the Install Key Validation web service. It is useful when you want to confirm that you have installed the InstallKey web application properly.

Although it is not a practical example of the usage of the ClientSideValidator, the source code for this tool is provided and includes working code examples for key validation, usage invalidation and reading custom properties.

Win Forms Sample

The Win Forms Sample application demonstrates how the ClientSideValidator can be used to restrict application access when the key has not been applied. If the application has not been validated, it will prompt the user to enter a key.

To set up this application, you will first need to create a keyset and a key using the InstallKey key management web application. After creating the keyset, modify the sample source code to insert the keyset values and validation URL into the code where defined.

Web App Sample

The Web App Sample application demonstrates how the ClientSideValidator can be used to restrict application access when the key has not been applied in an ASP.Net Web Application.

This sample also contains examples of these concepts.

- How to provide custom Client Side Identifiers
- How to store the surety in a database

To set up this application, you will first need to create a keyset and a key using the InstallKey key management web application. When creating the keyset, set the Minimum Identifier Matches to 1. After creating the keyset, modify the sample source code to insert the keyset values and validation URL into the code where defined in the web.config file.

To create the database that will be used to store the surety, view the Default.aspx page. This page contains the script that will create the database login and necessary table schema.

Pre Post Sample

The Pre Post Sample project provides working source code for providing custom code that is executed as part of the validation process. This sample shows how a Pre Validate method can force the validation to fail and shows how custom data can be included in the property bag to be sent to and from the server during validation. The Client Side Tester is also used with this sample to load and read values to and from the property bag.

See the chapter on Customizing the Validation Process for more information about how to add your custom code to the pre and post validation process.

Api Sample

The API Sample project provides working source code for the InstallKey API. This sample has working examples of every API call.

See the chapter on InstallKey Server API for more information about how to integrate and call the API methods and properties from your custom code.

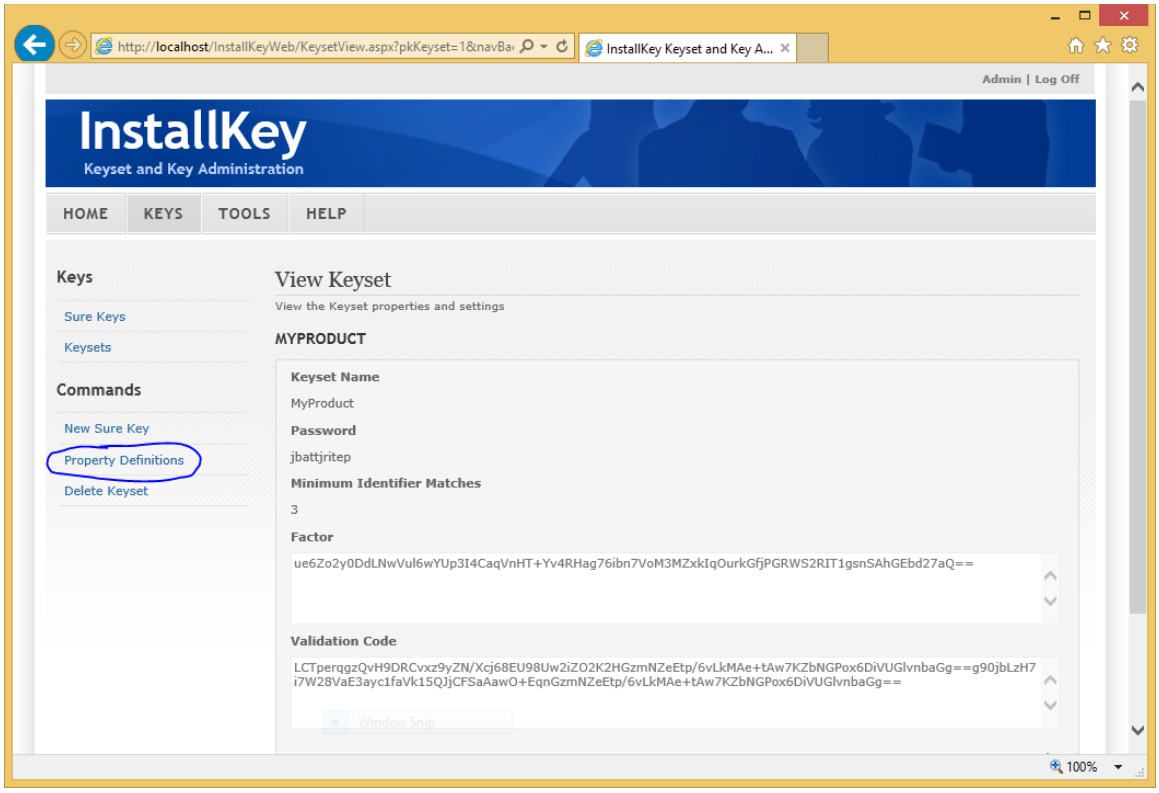
8. Custom Key Properties

Custom key specific properties can be sent securely from the server to the client by embedding the information in the surety. Because the properties are part of the surety, the same validations and methodology that prevents modifications and tampering to the surety ensures that the contents of the properties are also securely stored and not modifiable by the user.

Custom properties can be defined and maintained directly from within the InstallKey key management web application. No custom code needs to be added to the web application. After validation, the property values can be retrieved from the surety on the client.

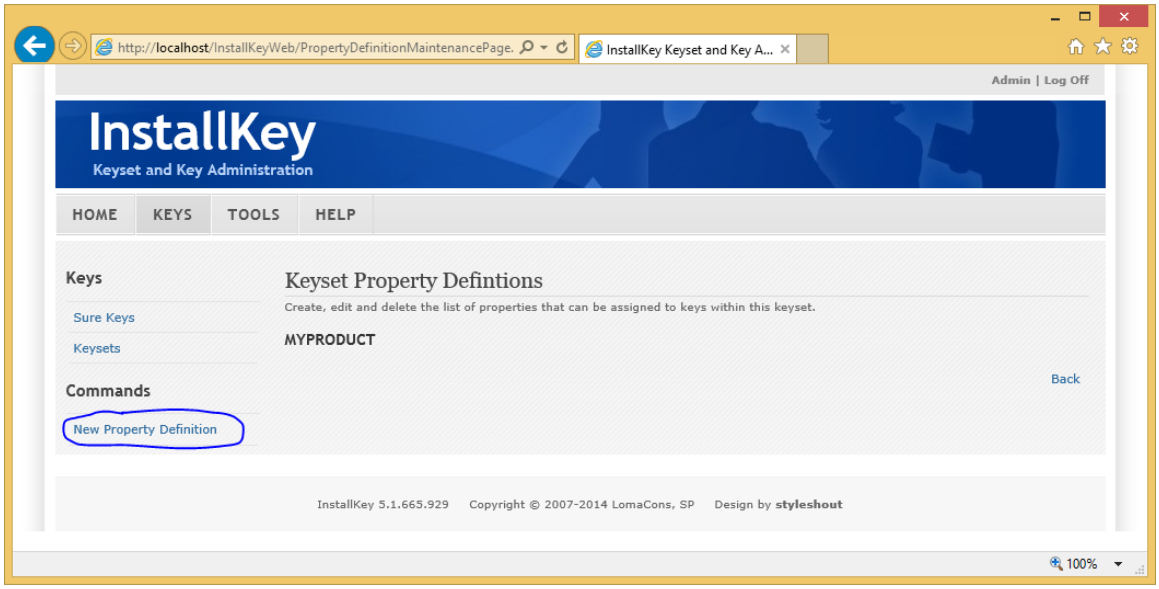
Defining Properties on a Keyset

To add a property to a key, the property must first be defined on the Keyset. Navigate to the View keyset page for the keyset that you want to add a property to, and click the Property Definitions Link.



Property Definitions Link

In the page that follows click the New Property Definition Link to navigate to the Add Property Definition page.



Location of the New Property Definition Link

On the Add Property Definition page, define the properties of the property definition. The Name and Grouping Name are used to uniquely identify the property on the keys and find them from within your client code. The other settings define how the property will appear and function on the Key Add and Key Edit web pages.

The Label Text is the value that will displayed on the key edit web page.

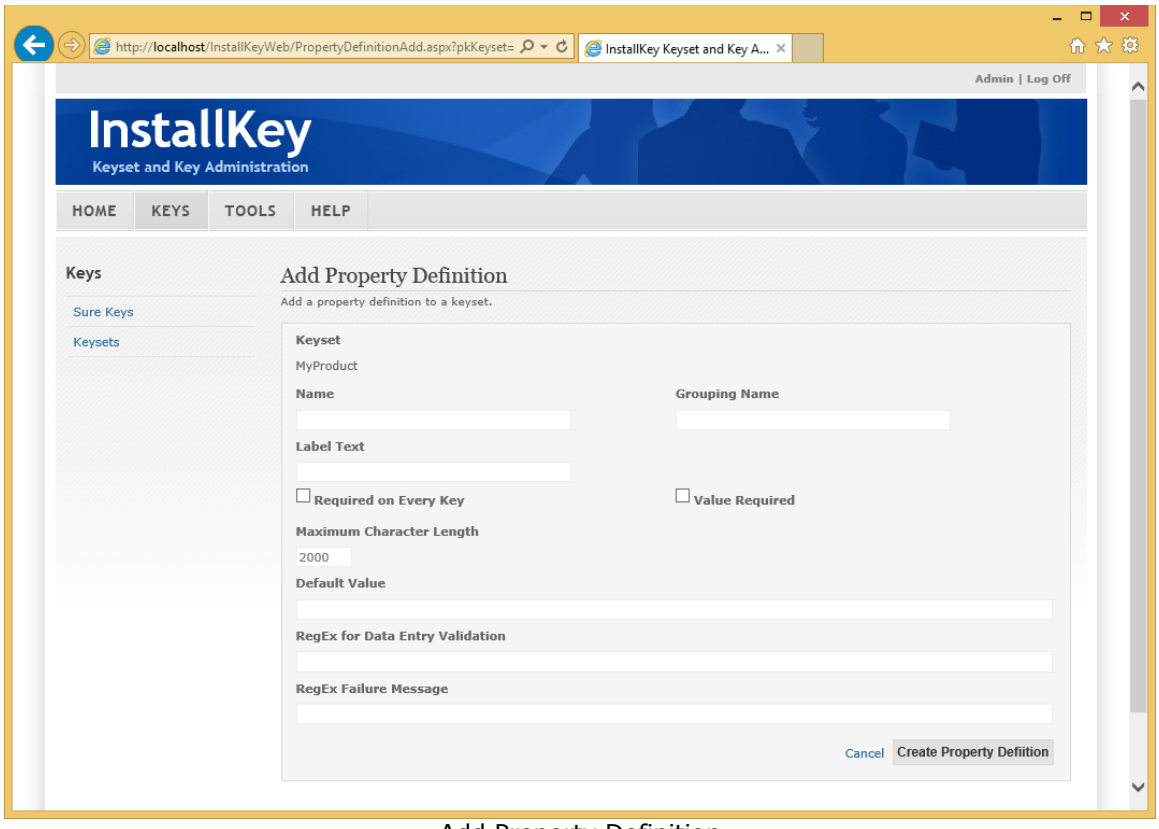
The Required on Every Key checkbox defines if the property must be assigned to every key in the keyset. If this checkbox is clear, then the property is optional on the keys.

The Value Required checkbox defines if a value must be entered for the property value.

The Maximum Character Length is the max allowed characters that the user can enter for the property value.

The Default Value is the value that will be prefilled in the key add web page when adding a new key.

The RegEx value will be used to validate the data entry on the key add and key edit web pages. The RegEx Failure Message is the text that will be displayed if the user does not enter a value that matches the regex expression.



Add Property Definition

As an example, suppose that you want to add a property to your keys to indicate that your customer has purchased the professional edition of your product. Then when your customer runs your application, the code can look for this property in the surety and respond accordingly.

To create the definition for such a property, enter these values in the property definition to define a property that holds a Yes or No value.

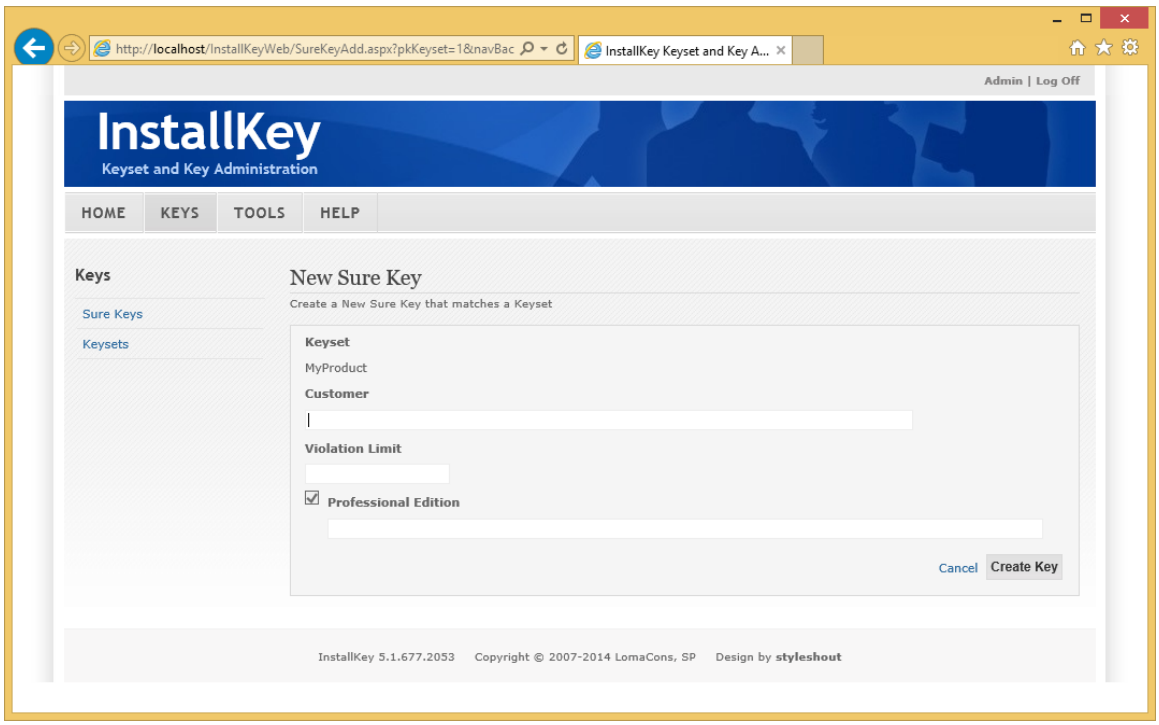
Name	IsProfessional
Grouping	MyProductFeatures
Label Text	Professional Edition
Max Char Length	3
Required on Every Key	Yes (checked)
Value Required	Yes (checked)
Default Value	
RegEx for Data Entry	^Yes|No$
RegEx Failure Message	Professional Edition must be Yes or No

Note that keyset property definitions can be added, edited or deleted from keysets at any time. However, it is important to understand the implications of changing the property definitions on the Keyset. Depending upon how you are using the properties and depending upon how your client code utilizes them, you can lose data or worse, your application may not function properly.

Assigning Properties to Keys

After a property definition has been assigned to a keyset, the property will be visible when adding or editing the keys for that keyset. The field label text and the data entry rules defined on the property definition are used to display a data entry field.

For example, the IsProfessional property definition defined above will appear as a label and data entry text box when editing a key. Because the property definition has a regular expression, the text is limited to Yes or No.



Key Data Entry with Custom Property

Retrieving Properties on the Client

The properties are available on the client any time after the key has been validated. Because the property values are stored securely in the surety, they are available without a connection to the server, and cannot be modified without invalidating the surety.

To retrieve the properties on the client, retrieve the values from the LocalSuretyContent object. An example of reading the IsProfessonal property is below.

```
public bool GetIsProfessional(ClientSideValidator csv){
   bool result = false;

   foreach(SuretyProperty sp in csv.LocalSuretyContent.Properties){
       if (sp.GroupName == "MyProductFeatures"){
           if (sp.Name == "IsProfessional" && sp.Value == "Yes"){
               result = true;
           }
           if (sp.Name == "IsProfessional" && sp.Value == "No"){
               result = false;
           }
       }
   }
   return(result);
}
```

Client Side Properties Sample

Refreshing the Property Values on the Client

Changing the property values on the client will require changing the editing the value stored in the key on the server and then refreshing the client. Because the properties are stored securely in the surety, the

values can only be changed by calling the ValidateAgainstServer method to refresh the surety with the current content.

Using the example above, assume that your customer first purchased the standard edition of your product and a few weeks later the customer purchased an upgrade to the professional edition. To effect this change, you will edit the customer's key, changing the value of the Is Professional property from No to Yes. Then the customer will refresh the surety on the client side to expose your product's additional functionality.

9. Storing the Local Surety

Once the key has been validated against the server, the surety is persisted locally so that it can be used to validate without another round trip to the server. By default the surety is persisted as an XML file in the current working folder. Two implementations of this interface are included with InstallKey and a third is provided as sample.

The ISuretyStorage Interface

The ISuretyStorage is used by the ClientSideValidator object to persist and read the local surety. Any object that implements ISuretyStorage can be used to persist and read the surety. This provides unlimited possibilities for implementing a custom storage mechanism.

FileSuretyStorage

This is the default Surety Storage object used by InstallKey. The FileSuretyStorage object is used to persist and read the surety from a file on the file system. Below is a code snippet showing how to create a ClientSideValidator object that will persist the surety to a file in the root of the C drive.

```
ClientSideValidator cl;
ISuretyStorage storage;

storage = new FileSuretyStorage(@"C:\Surety.xml");
cl = new ClientSideValidator (_password, _keysetFactor);
cl.SuretyStorage = storage;
```

SuretyStorageIsolated

The SuretyStorageIsolated object is used to persist the surety as an XML file using the .Net isolated storage mechanism. By default the xml file is stored using GetMachineStoreForDomain scope. Below is a code snippet showing how to create a ClientSideValidator object that will persist the surety to a file in the root of the Isolated Storage.

```
ClientSideValidator cl;
ISuretyStorage storage;

storage = new SuretyStorageIsolated();
storage.FilePath = "Surety.xml"
cl = new ClientSideValidator ( password,  keysetFactor);
cl.SuretyStorage = storage;
```

Database Storage

Another surety storage possibility is to store the surety in a database record. The WebAppSample includes an example of how to implement an ISuretyStorage such that the surety can be persisted to/from a database.

Securing the Surety Content

Regardless of where the surety is stored, it can be casually viewed. In the case of storing the surety in a file, the contents can be viewed with any text editor. This is just the nature of XML, and it is also the intention of InstallKey to allow the surety to be viewed. Because the surety is tamperproof and secure, the user will invalidate the surety if they attempt to modify it. Therefore, it may be best to allow the user to see the content just to show what information is collected and prove that you have nothing to hide.

However, you may not want the user to be able to see the contents. In this case, set the EncryptLocalSurety property to true. When EncryptLocalSurety is true, the entire contents of the surety will be encrypted when the surety is stored on the local system.

10. Validation without an Internet Connection

There are scenarios where a customer may not be able to validate a sure key because access to the internet is not available. To facilitate this, InstallKey provides a way to perform secure validation.

Use Case and Solution

What happens when your customer wants to install your application in a highly secure, or locked down environment where the internet cannot be connected to for any reason? In this scenario, the web service call will always fail, and the sure key cannot be validated directly.

The solution to this is a manual transport that maintains the security of the validation process. To do this these steps can be taken.

1. The un-validated surety is exported as an xml file on the local machine
2. The surety file is copied to a floppy disk or USB key.
3. The surety file is taken to a machine that does have internet access.
4. The surety file is uploaded to the InstallKey web server and is validated.
5. The validated surety file is downloaded from the InstallKey web server.
6. The validated surety file is copied to a floppy disk or USB key.
7. The validated surety file is taken to the machine that does not have internet access.
8. The validated surety file is applied, and the application is valdiated.

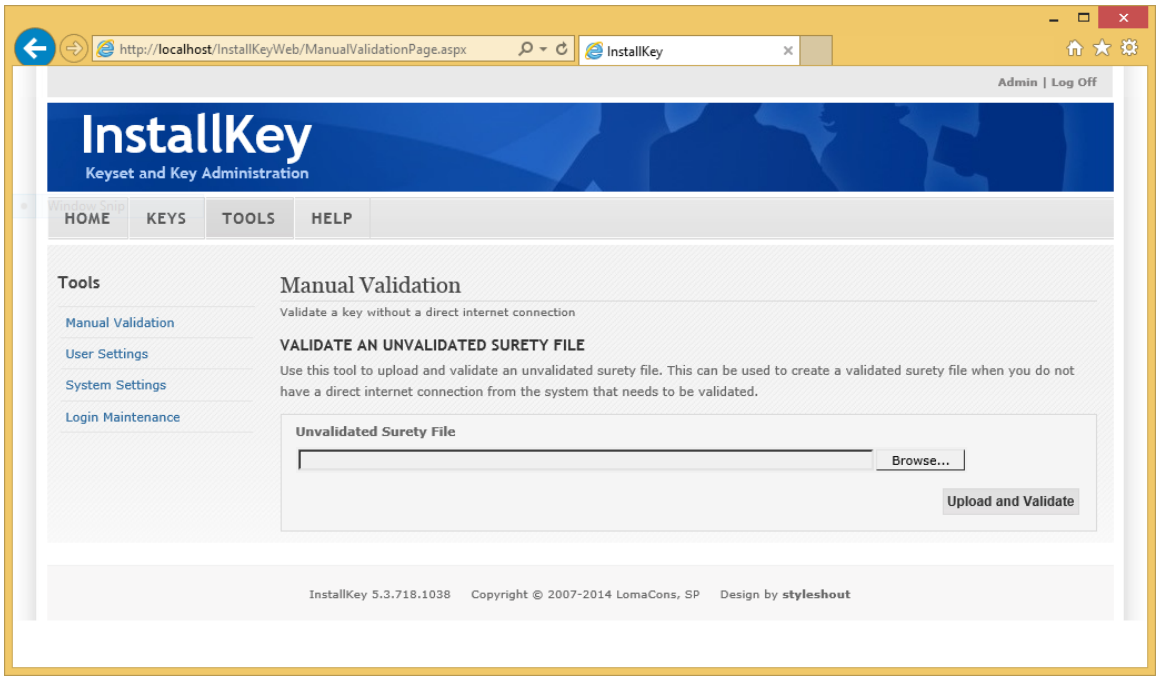
All subsequent validations will not require another round trip to the server as all the necessary information is already securely in the surety stored on the local machine.

Security of the Solution

This manual method of validating the sure key is just as secure as validating it directly over the internet. Both the un-validated and validated sureties are tamperproof, and are also specific to the machine where they are installed. This makes it virtually impossible to modify the surety, and therefore virtually impossible to use the validated surety file on more than one machine.

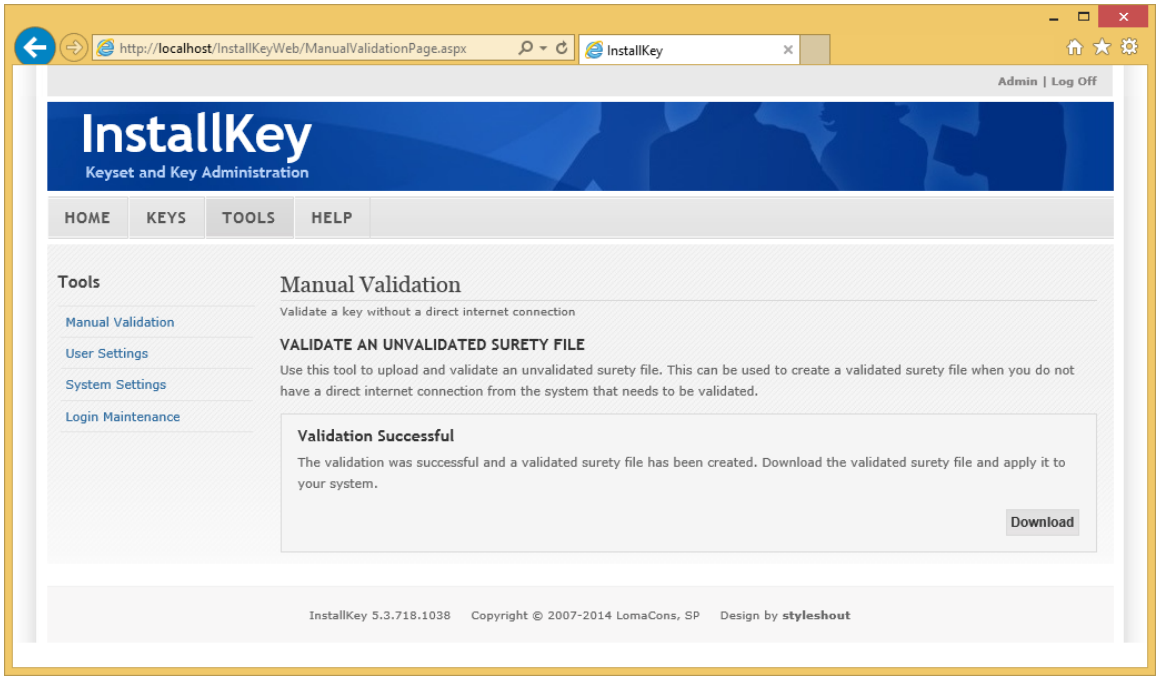
Validation Web Page

To validate the surety upload the un-validated file to the Manual Validation web page.



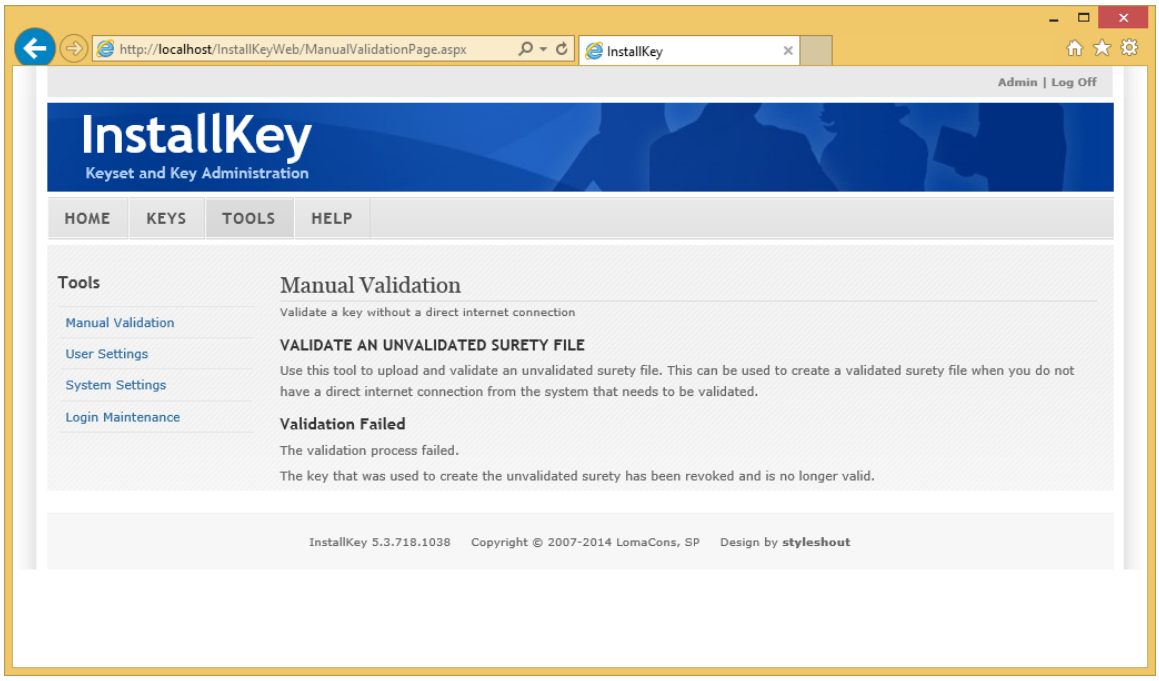
Manual Validation Web Page

The file will be validated against the server (just as it would during the ValidateAgainstServer method call) and if all is well, a validated surety file will be created and you will be prompted to download the file.



Validated File Successfully – Ready To Download

If the validation process fails (because the surety was tampered with, the key is invalid or revoked, etc.) then the Validation page will display an error message to the user. Because of the error, a validated surety file will not be created.



Manual Validation Failure

Example

See the Client Side Tester sample for an example of how to export the un-validated surety as a file, validate the surety and apply the validated surety.

11. Customizing the Validation Process

InstallKey provides all the necessary methods and properties to handle common key validation needs such as trial validations, properties and revoking keys. However, there are an unlimited number of custom scenarios that InstallKey cannot provide, but can be easily implemented using custom code. Ultimately, this provides a very powerful method to customizing the validation process to fit any need, without having to rebuild or modify the provided InstallKey code.

For this reason, pre-validation and post-validation interfaces are provided such that you can add your custom code into the validation process. In addition, a property bag is provided that allows your custom code to send any serializable object securely to and from the server as part of the surety.

The Validation Process

The steps below outline the step by step process that occurs when the ValidateAgainstServer method is called, and where your custom pre validate or post validate code will be called.

- The ValidateAgainstServer method is called on the client. Optionally a set of property bag values can be sent to the server along with the 30 character key.
- The surety containing the key, identifiers and property bag is sent to the server.
- Once on the server, the surety is validated to confirm that it was not tampered with.
- The key provided is confirmed that is a match for the keyset and that it exists in the InstallKey database. If the key does not exist in the database, the validation process ends and the result is immediately sent back to the client.
- The key is checked to see if it was revoked. If the key is revoked, the validation process ends and the result is immediately sent back to the client.
- The number of identifiers provided in the surety is checked against the minimum number of identifiers required for the keyset. If there are not enough identifiers, validation process ends and the result is immediately sent back to the client.
- PreValidate – If the pre-validate assembly is defined, the assembly is loaded and the custom pre-validate code is executed. A reference to the property bag is passed to the pre-validate method and may be modified by the custom code. If the prevalidate method indicates that the validation process should fail at this point, the validation process ends and the result is immediately sent back to the client.
- The identifiers provided by the surety are checked against any identifiers that were already recorded in the database for the key during any prior successful validations. If the key has been validated too many times, the validation process ends and a violation result is immediately sent back to the client.
- PostValidate – If the post validation assembly is defined, the assembly is loaded and the custom post-validation code is executed. A reference to the property bag is passed to the pre-validate method and may be modified by the custom code.
- The property bag, and any other custom properties are added to the surety, and surety is sent securely back to client with a successful status.
- Back on the client, the validated surety is validated to confirm that it was not modified or tampered with, stored locally on the client and will contain all surety content, properties and the contents of the property bag.

Create and Install a Custom Code Assembly

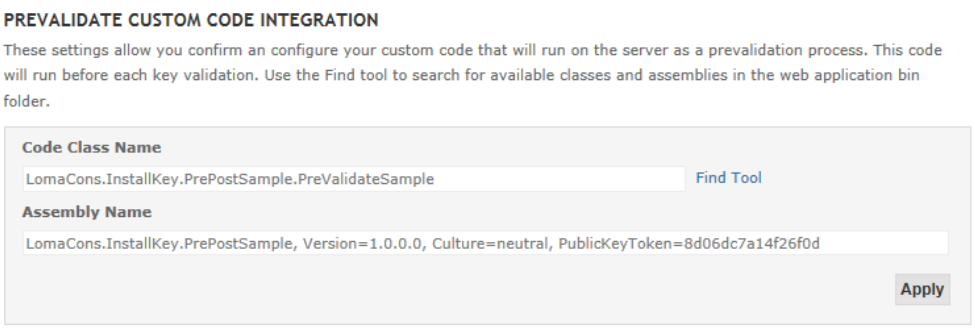
Follow these steps to create a Pre Validate assembly and associate it with an installation of InstallKey. The steps below are for the Pre Validate, however, the process is the same for Post Validate. A working sample is provided in the PrePostSample project included with the InstallKey distribution zip file.

- Open Visual Studio and create a new Class Library project
- Add a reference to the LomaCons.InstallKey.Common.dll assembly
- Add a strong name key to the project, and apply the strong name to the project
- Create a class object that implements the IPreValidate interface. A code sample is pictured below.

```
public class PreValidateSample : IPreValidate {

    public PreValidateSample() {
        return;
    }

    public EnumPreValidationResult PreValidate(
            string keysetName, string sureKey,
            Guid installGuid, IDictionary<string, object> propertyBag) {

        // your custom code goes here

        return(EnumPreValidationResult.Continue);
    }
}
```

PreValidate Code Sample

- Build the assembly into a .dll file.
- Copy the .dll file into the bin folder of the InstallKey web application.
- Log into the InstallKey web application and navigate to the System Settings page. On the system settings page there is a section to define a custom code integration.



Pre Validate Configuration Settings

- Enter the full code class name and assembly name that implements the IPreValidate interface. Or use the Find Tool to locate the values. The find tool will search the bin folder of any dll file that implements the IPreVadiate interface and will auto populate the text fields. Click the Apply button to save the settings.
- If there are no errors, the code is properly installed and will be executed during validations. Otherwise, an error message will be displayed if there was any errors while configuring the custom code interfaces.

The PreValidate Method

When the PreValidate method is called during the validation process, information about the key being validated is passed to the code in your custom assembly that implements the IPreValidate interface. A description of the parameters and return code is below.

```
public interface IPreValidate {
  EnumPreValidationResult PreValidate(
         string keysetName,
         string sureKey,
         Guid installGuid,
         IDictionary<string, object> propertyBag);
  }
```

IPreValidate Interface Definition

keysetName
The keysetName parameter will contain the name of the keyset that corresponds to the key being validated. This parameter is useful if you have multiple keysets and need to have different processing or code paths that are dependent upon the keyset.

sureKey
The sureKey parameter will contain the key being validated in the human readable, 30 character, key value form with dashes.

installGuid
The installGuid parameter will contain the unique guid value that corresponds to the key being validated. This value can be used to query the database or otherwise locate additional information about the key being validated. See the PrePostSample code for a working code sample that reads content from the InstallKey database.

propertyBag
The property bag is a dictionary collection that is passed securely from the client to the server. This allows you to pass additional values and serializable objects securely to and from the server.

If a property bag was provided when the ValidateAgainstServer method was called, the property bag will contain the values provided from the client.

The key value pair items in the property bag can also be modified, added to or removed from within you custom code. The property bag is then sent back to the client and the values contained can be read from the local surety on the client.

return code
The return code is an EnumPreValidationResult value, and is used to indicate if the validation process should continue or fail. This is useful in situations where you want you custom code to evaluate some condition and optionally fail the validation process.

If the return code is set to EnumPreValidationResult.Fail, the validation process will end without fully validating the key. On the client, the ValidateAgainstServer() method call return EnumValidateResult.PreValidateFailure value accordingly. This is useful when you need to have your client side code handle your failure.

If the return code is set to EnumPreValidationResult.Continue, the InstallKey validation process continues normally.

The PostValidate Method

When the PostValidate method is called during the validation process, information about the key being validated and its usage record is passed to the code in your custom assembly that implements the IPostValidate interface.

```
public interface IPostValidate {
  void PostValidate(
         string keysetName,
         string sureKey,
         int pkUsageValidation,
         IDictionary<string, object> propertyBag);
  }
```

IPostValidate Interface Definition

keysetName
The keysetName parameter will contain the name of the keyset that corresponds to the key being validated. This parameter is useful if you have multiple keysets and need to have different processing or code paths that are dependent upon the keyset.

sureKey
The sureKey parameter will contain the key being validated in the human readable, 30 character, key value form with dashes.

pkUsageValidation
The pkUsageValidation parameter will contain the primary key of the UsageValidation record that represents the key being validated. From this primary key, other key and keyset information can be queried.

propertyBag
The property bag is a dictionary collection that is passed securely from the client to the server. This allows you to pass additional values and serializable objects securely to and from the server.

If a property bag was provided when the ValidateAgainstServer method was called, the property bag will contain the values provided from the client.

The key value pair items in the property bag can also be modified, added to or removed from within you custom code. The property bag is then sent back to the client and the values contained can be read from the local surety on the client.

Adding Data to the Database

Custom values and data may be stored using any method or location of your choosing, including storing them directly in the InstallKey database.

Any modifications that you make to the InstallKey database schema is always made at your own risk and will always contain the possibly that your changes may not work with future releases of InstallKey. However, the methodology that follows is least likely to cause issues with future changes to InstallKey, and is the preferred method to add you own content to the database.

When storing custom values in the InstallKey database, the recommended method is to store these attributes in a separate table that has a one-to-one foreign key relationship with the InstallKey table. The T-SQL code snippet below shows how to create a table to hold a custom property called CustomerEmail. This will be used in the following use case and sample.

```
CREATE TABLE dbo.CustomKeyAttr(
   PKCustomKeyAttr int NOT NULL,
   CustomerEmail nvarchar(100) NOT NULL,
   CONSTRAINT PK_CustomKeyAttr PRIMARY KEY CLUSTERED (
       PKCustomKeyAttr ASC
   )
)

ALTER TABLE dbo.CustomKeyAttr
ADD CONSTRAINT FK CustomKeyAtt SureKey
FOREIGN KEY (PKCustomKeyAttr)
REFERENCES dbo.SureKey (PKSureKey) ON DELETE CASCADE
```

Custom Data Table in InstallKey Database

Sample Use Case

As an example, suppose that you want to have the user provide their email address and have the email address sent to the server as part of the validation process. This value can be added as a property bag value, sent to the server and stored in a custom database table.

The code below will put an email value into the property bag. The property bag will be included in the surety that is sent securely to the server.

```
EnumValidateResult Validate(ClientSideValidator cv,
                                          string keyValue, string emailAddress) {
   EnumValidateResult result;
   IDictionary<string, object> initalPropertyBag;

   initalPropertyBag = new Dictionary<string, object>();
   initalPropertyBag.Add("EMailAddress", emailAddress);
   result = cv.ValidateAgainstServer(keyValue, initalPropertyBag);
   return(result);
}
```

Add Content to the Property Bag before Validation

The code below is from a PostValidate implementation that will read the content from the property bag, and store the value in a custom database table. The code below assumes that you have created a custom table using the SQL code above.

```
public class CustomPostValidate : IPostValidate {

  public CustomPostValidate() {
      return;
  }

  public void PostValidate(string keysetName, string sureKey,
                int pkUsageValidation, IDictionary<string, object> propertyBag) {
      int pkSureKey;

      if (propertyBag.ContainsKey("EMailAddress") == false)
          return;
      pkSureKey = GetPrimaryKeyOfSureKeyFromUsageValdiation(pkUsageValidation);
      PersistEMailAddress(pkSureKey, propertyBag["EMailAddress"].ToString());
      return;
  }

  private void PersistEMailAddress(int pkSureKey, string emailAddress) {
      SqlConnection conn;
      SqlCommand cmd;
      StringBuilder sb;

      sb = new StringBuilder();
      sb.Append(@"IF EXISTS (SELECT * FROM dbo.CustomKeyAttr ");
      sb.Append(@"WHERE PKCustomKeyAttr = @pkSureKey) ");
      sb.Append(@"BEGIN ");
      sb.Append(@"UPDATE dbo.CustomKeyAttr SET CustomerEmail = @emailAddress ");
      sb.Append(@"WHERE PKCustomKeyAttr = @pkSureKey ");
      sb.Append(@"END ELSE BEGIN ");
      sb.Append(@"INSERT INTO dbo.CustomKeyAttr (PKCustomKeyAttr, CustomerEmail) ");
      sb.Append(@"VALUES (@pkSureKey, @emailAddress) ");
      sb.Append(@"END");
      conn = new  SqlConnection(
                 ConfigurationManager.ConnectionStrings["InstallKey"].ConnectionString);
      using (conn) {
          conn.Open();
          cmd = new SqlCommand();
          using (cmd) {
              cmd.Connection = conn;
              cmd.CommandType = CommandType.Text;
              cmd.CommandText = sb.ToString();
              cmd.Parameters.Add(new SqlParameter("@pkSureKey", pkSureKey));
              cmd.Parameters.Add(new SqlParameter("@emailAddress", emailAddress));
              cmd.ExecuteNonQuery();
          }
          conn.Close();
      }
      return;
  }

  private int GetPrimaryKeyOfSureKeyFromUsageValdiation(int pkUsageValidation) {
      SqlConnection conn;
      SqlCommand cmd;
      object result;
      StringBuilder sb;

      sb = new StringBuilder();
      sb.Append(@"SELECT ku.FKSureKey as PKSureKey ");
      sb.Append(@"FROM dbo.KeyUsage as ku INNER JOIN ");
      sb.Append(@"dbo.UsageValidation as uv on uv.FKKeyUsage = PKKeyUsage ");
      sb.Append(@"WHERE uv.PKUsageValidation = @pkUsageValidation");
      conn = new SqlConnection(
                 ConfigurationManager.ConnectionStrings["InstallKey"].ConnectionString);
      using (conn) {
          conn.Open();
          cmd = new SqlCommand();
          using (cmd) {
              cmd.Connection = conn;
              cmd.CommandType = CommandType.Text;
              cmd.CommandText = sb.ToString();
```

```
            cmd.Parameters.Add(
                        new SqlParameter("@pkUsageValidation", pkUsageValidation));
            result = cmd.ExecuteScalar();
        }
        conn.Close();
    }
    return (Convert.ToInt32(result));
  }

}
```

Post Validate Implementation

After the validation has completed, the client side code below reads the values from the property bag after validation and shows a result to the user. This code can be called any time after validation because the value is stored securely in the local surety.

```
public void ShowEMailAddress(ClientSideValidator cv){
  if (cv.IsValidated == false)
      return;
  if (cv.LocalSuretyContent.PropertyBag.ContainsKey("EMailAddress") == false)
      return;
  MessageBox.Show(cv.LocalSuretyContent.PropertyBag["EMailAddress"].ToString());
  return;
}
```

Read Content from Property Bag after Validation

12. Demos and Expiring Trials

Trial Validation is done on a key specific basis where an optional trial expiration date can be assigned to a key. The expiration date can be securely validated anytime, using the same key validation methods.

Defining the Trial Expiration Date

On the Sure Key Edit page or the Add Sure Key page, the date that the trial expires can be assigned to the key using the Trial Expires field. If the field is left blank, the key will not expire.

Trial Use Case

The use case that follows is similar to the use case presented in the prior chapters. In this use case the sure key is evaluated and the trial expiration date is compared to the current date using the IsTrial and TrialExpires properties of the validated surety. The distribution is considered to be valid if the surety is both valid and the trial has not expired.

```
private const string _keysetName = "...";
private const string  password = "...";
private const string _factor = "...";
private const string _validationCode = "...";
private const string _validationUrl = @"http://.../InstallKeyWeb/Validator.svc";

/// <summary>
/// Validates that the application is validated.  Will validate against
/// the server if the local surety file is not found or is not valid
/// </summary>
/// <returns>
/// true if the surety and trial date are valid, false otherwise
/// </return>
public bool Validate() {
  ClientSideValidator validator;
  string key;
  EnumValidateResult result;

  // create an instance of the ClientSideValidator object
  validator = new ClientSideValidator(_keysetName, _password, _factor, _validationCode);
  validator.SetServiceAddress(_validationUrl);
  // check to see if the surety already exists and is valid
  if (validator.IsValidated) {
      // already validated - now check for expired
      return(validator.IsTrialExpired(EnumIsTrialMethodology.ServerThenLocal));
  }
  key = PromptUserForKey();
  // check to see if the user typed in the key correctly
  if (validator.DoesKeyMatchKeyset(key) == false)
      return (false);
  // validate the key against the server
  result = validator.ValidateAgainstServer(key);
  // check the result
  if (result == EnumValidateResult.OK) {
      // validated - now check for expired
      return(validator.IsTrialExpired(EnumIsTrialMethodology.ServerThenLocal));
  }
  return (false);
}
```

Trial Expires Date Sample

Evaluating the Expiration Status

The IsTrialExpired method is used to evaluate the trial expiration date based on the trial date stored in the surety and the current date. The source of the current date used for the comparison is defined by the methodology enumeration parameter.

Local Only – the date in the surety is compared to the local system date.

Server Only – the date in the surety is compared to the date as provided by the server. For this to work, the system must be able to connect to the validation web service. Because the date is compared against the date on the server, it will not be possible for the user to circumvent the expiration date by changing the current date on the local machine.

ServerThenLocal – the date in the surety is compared to the date on the server. If there is an exception when retrieving the date from the server, the local time will be used instead.

When using the date from the server, date is provided by a call to the Validator.svc web service. By using the date from the server (rather than the local system date) it is harder for the user to bypass an expired date by changing the local system date. However, this means that the user must have an internet connection in order to evaluate the trial status.

Attempting to Bypass the Trial Date

When using the Local Only setting, the easiest way to bypass an expired trial date, is for the user to change the system date. For this reason it is important not to rely exclusively on the local time.

The server time is more difficult to bypass. Because the current date is from the server, the user cannot change it. Similar to the validation surety, the date returned from the server contains a validation code that is used to validate that the date was not tampered with. Attempting to modify the date or the validation code will cause the date to become invalid.

Expiration Timespan

Another variation on the trial dates would be to define a date that expires "x" number of days, weeks or months from the first validation or usage. A future release of InstallKey will include this functionality. However, this can be implemented today by writing some custom code to extend the functionality of InstallKey to fit your needs.

To implement this, you will need to understand how to extend InstallKey and a basic understanding of the database relationships. You should review and understand the chapters on adding Custom Key Properties and Customizing the Validation Process before implementing the expiration timespan.

Using one or more of the following guidelines and ideas, you can easily add this functionality to your implementation of InstallKey.

- Create a custom key property to indicate that the key expires on a timespan.
- Create a PostValidate method. In this post validate method determine if the key expires on a timespan. If it does, calculate the expiration date from the first usage date and put the expiration date in the surety property bag.
- In your client code, after calling the IsValidated property, retrieve the expiration date from the surety property bag, and determine if the time has expired.
- To retrieve the custom key property from within your PostValidate method, execute a query against the InstallKey database using the pkUsageValidation parameter provided by the post validate calling code. By joining the UsageValidation, KeyUsage, SureKeyProperty and PropertyDefinition tables, you can determine if the key contains your custom property. A sample query is below.

```
SELECT
    skp.Value
FROM
    dbo.UsageValidation AS uv
    INNER JOIN
    dbo.KeyUsage AS ku ON uv.FKKeyUsage = ku.PKKeyUsage
    INNER JOIN
    dbo.SureKeyProperty AS skp ON ku.FKSureKey = skp.FKSureKey
    INNER JOIN
    dbo.PropertyDefinition AS pd ON skp.FKPropertyDefinition = pd.PKPropertyDefinition
WHERE
    uv.PKUsageValidation = @pkUsageValidation
    AND
    pd.Name = @propertyName
```

Sample Custom Property Query

- To retrieve the first usage date for a unique validation from within your PostValidate method, execute a query against the UsageValidation table using the pkUsageValidation parameter.

```
SELECT
    MIN(uv.ValidationDateUtc) as FirstUsageValidationDateUtc
FROM
    dbo.UsageValidation as uv
WHERE
    uv.FKKeyUsage IN (
        SELECT FKKeyUsage
        FROM dbo.UsageValidation
        WHERE PKUsageValidation = @pkUsageValidation
    )
```

Sample First Usage Query

Keyless Trial

Another variation on the expiring timespan is a keyless trial, where the distribution expires even if the user removes and re-installs the distribution.

The use case for a keyless trial is one where you freely give away your distribution, and the distribution expires “x” number of days after it is first used unless the user obtains a key. In this scenario, the person who uses your software distribution does not need to obtain and enter a key before using the software distribution.

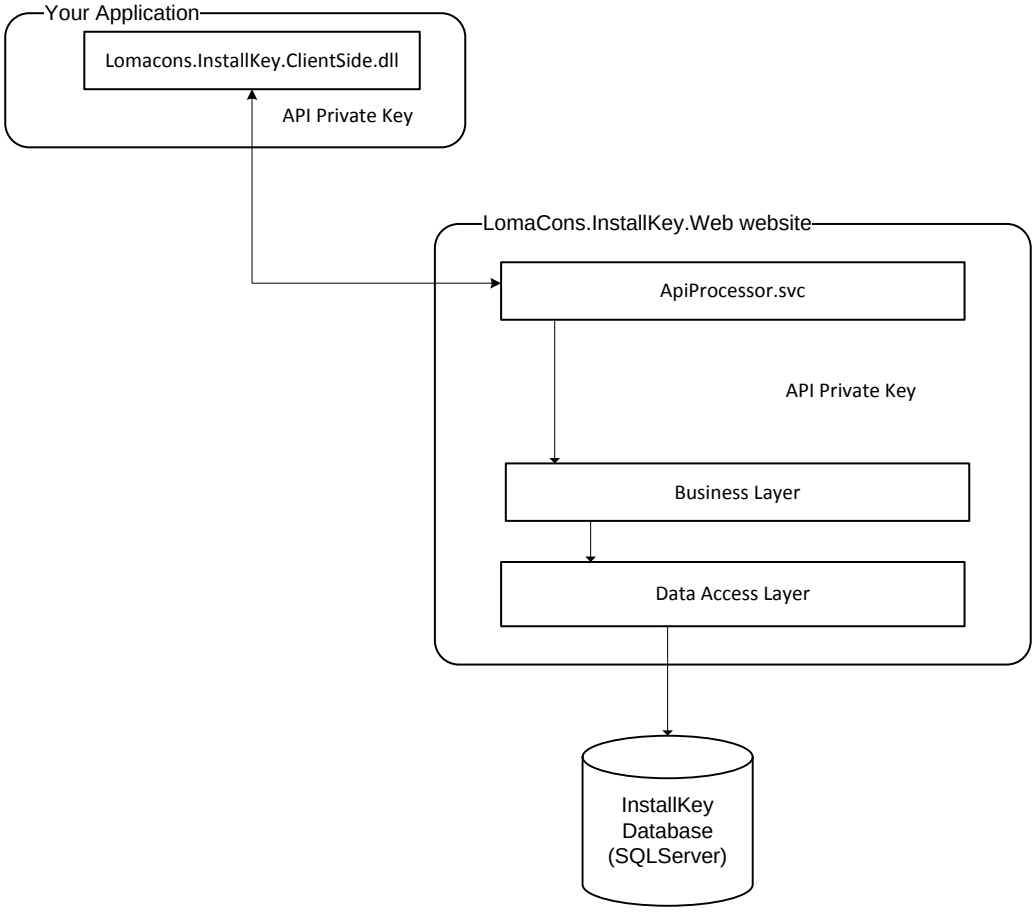
However, to implement this, a key still needs to be validated in order to securely record the unique system identifiers with the InstallKey server. Follow these ideas and guidelines to implement this “transparent validation” by extending InstallKey with custom code to meet your business needs.

- Implement an Expiring Timespan using the methodology described above.
- Create a sure key with an expiring timespan using the Key Management web application, or API. This will be the “transparent key” The user will not need to see the key or be aware of the existence of this key.
- Use the ClientValidator in the distribution to validate just as you would any other key that you generate and provide to a user. But, instead of prompting the user to enter a sure key – validate using the transparent sure key. This 30 character sure key can be hard coded, or you may want to put this key in the app.config file. But wherever you store the key, the user should not need to enter the key.
- Provide a way for the user to enter a key to unlock the software.

13. InstallKey Server API

An API will allow programmatic manipulation of the keys and keysets from your custom applications written in C# or Visual Basic .Net. (Although all the samples and descriptions that follow are written in C#, the API can be called from VB.Net just as well.) The most common usage is to be able to programmatically create keys. To use the API, you will need your API Private Key and a Login with the API Access rights.

The diagram below shows the components and interactions of the InstallKey API. The API is implemented as an http based WCF web service that allows your application to connect and update the content from any system that has access to the web service endpoint. The descriptions that follow describe the components in more detail.



InstallKey API Architecture

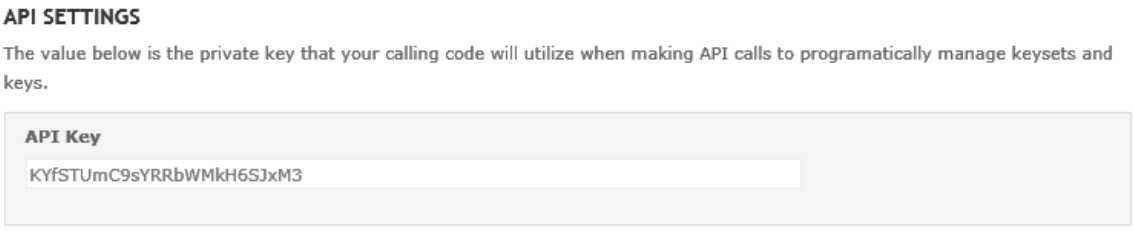
Your Application

Your application will reference the InstallKey Client Side dll assembly. The Client Side assembly encapsulates the API calls and provides easy to use methods that hide the complexity of calling the API web services. The API can be called from C# or Visual Basic .Net.

API Key

The API key is a private key that is known by the InstallKey server and your calling code. It is not transmitted over the web service connection. However, this private key is used to encrypt a validation code that is used to confirm that the content sent to and from the server was not modified.

Every instance of the InstallKey Web application will have a unique API Key. This key can be found by viewing the System Settings on InstallKey web application. This value is uniquely generated and does not change.



API Settings viewed from the System Settings

LomaCons.InstallKey.Web website
The InstallKey website is the host for the ApiProcessor web service. This ApiProcessor web service is responsible for validating the request, processing the request and returning a response to the caller. Your code will not call this web service directly. Rather, the code in the client side assembly will negotiate these details.

Business Layer and Data Access Layer
The business layer and data access layers contain the core logic for creating and managing keys. It is used by both the API Processor and the InstallKey web application to provide good code reuse and to ensure consistency.

Security of the API

The API transmits data to and from the server in the clear accompanied by a validation code. This validation code ensures that the API calls and content is not tampered with as it moves from the caller to the server and back. Any attempt to modify the content will result in an error and a .Net exception will be thrown.

Although the data cannot be modified and is safe from tampering, the content sent to and from the web service can be seen if someone is viewing the raw content as it travels across a network. If your connection to the API is internal, on the same machine, or within an intranet environment, then transmitting the data in the clear is not likely to be an issue. However, if your connection to the API service is over the internet, you may want to consider running the ApiProcessor service over HTTPS. Calling the API methods over a secure channel will ensure that the content cannot be read or viewed.

HTTPS configuration and setup is covered in the installation chapter above, and must be enabled in the web.config before the API methods can be called over HTTPS. To configure your client code for https, you call the Api constructor passing in the URL to the API service. The API constructor will automatically detect if the URL scheme is http or https and adjust the client endpoint binding accordingly.

Sample

The following is a simple code snippet of how to create an instance of the Api object and create a key using C#.

```
// this method creates a surekey and returns the readable key value
public string CreateKey() {
  // local variables
  Api api;
  ApiKeyset keyset;
  int pkSureKey;
  ApiSureKey sureKey;

  // Create an instance of the Api object.  The url, private key and login are for this
  // sample - the real values that work in your environment will be different.
  api = new Api(@"http://localhost/LomaCons.InstallKey.Web/ApiProcessor.svc",
                                  "KYfSTUmC9sYRRbWMkH6SJxM3", "admin", "Xyz6284$");

  // get the keyset object for the MyFirstKeyset keyset (again, your keyset name may
  // be different)
  keyset = api.ReadKeyset("MyFirstKeyset");

  // create a sure key for a new customer with a violation limit of four
  pkSureKey = api.CreateSureKey(keyset.PKKeyset, "George Herman Ruth", 4, null, null);

  // get the sure key object for the newly created key
  sureKey = api.ReadSureKey(pkSureKey);

  //return the 30 character readable key
  return (sureKey.KeyValue);
  }
```

A fully functional, sample web application is included as an example of how the various method calls can be used to view, retrieve update and delete InstallKey content. This sample called ApiSample contains working examples of every API call.

Api Object

Before any methods can be called, an instance of the Api object must be created. To create the Api object, call one of the constructors, providing the service information, api private key, and login information.

Constructors

```
public Api(string serviceUrl, string apiKey, string loginName, string password)
```

This constructor will create an instance of the Api object. The serviceUrl is url of the ApiProcessor.svc web service endpoint. The apiKey is the private key for the Api as defined by the system settings. The key will be unique to the installation of InstallKey. The login name and password is the login that will be used by the service and it must have the Allow Api access right.

```
public Api(Uri serviceUrl, string apiKey, string loginName, string password)
```

This constructor will create an instance of the Api object, and allows you to pass the Url as a Uri object.

```
public Api(Binding binding, EndpointAddress endpointAddress, string apiKey, string
loginName, string password)
```

This constructor will create an instance of the Api object and allows you to define a custom binding and endpoint.

Api Properties and Methods

The following is a brief list and description of the Api Properties and methods.

Server Version

```
Version ServerVersion
```

This property will retrieve the version of the InstallKey server.

Read Keyset

```
ApiKeyset ReadKeyset(int pkKeyset)
```

This method is used to retrieve the Keyset and its Property Definitions for the keyset identified by the primary key provided. It will return an ApiKeyset object.

```
ApiKeyset ReadKeyset(string keysetName)
```

This method is used to retrieve the Keyset and its Property Definitions for the keyset identified by the keyset name provided. It will return an ApiKeyset object.

Read Keysets

This method returns a collection of ApiKeyset objects. The collection will not contain Property Definitions. To retrieve the Property Definitions, call the ReadKeyset() method for the individual keyset.

```
IList<ApiKeyset> ReadKeysets(KeysetFilter filter, KeysetSort sort, int skip, int limit)
```

This method is used to query for a list of keysets as defined on the server. Filter is used to describe the records to search for. Sort defines the sort order of the values returned. Skip and limit are used to define the subset of objects that match the filter and sort.

```
IList<ApiKeyset> ReadKeysets(IEnumerable<KeysetFilter> filters,
                                IEnumerable<KeysetSort> sorts, int skip, int limit)
```

This method is used to query for a list of Keysets using multiple filters and sorts. The filters will be combined to produce an “AND” style query. The sorts will be applied in the list order. Skip and limit are used to define the subset of objects that match the filter and sort.

Read Keysets Count

```
int ReadKeysetsCount(KeysetFilter filter)
```

This method will return the number of records that match the filter.

```
int ReadKeysetsCount(IEnumerable<KeysetFilter> filters)
```

This method will return the number of records that match the filters.

Read Sure Key

```
ApiSureKey ReadSureKey(int pkSureKey)
```

This method is used to retrieve the Sure Key and its Properties for the sure key identified by the primary key provided. It will return an ApiSureKey object.

Read Sure Keys

The objects in the collection will not contain any properties, even if there are properties on the sure key. To retrieve the Properties, call the ReadSureKey() method for the individual sure key.

```
IList<ApiSureKey> ReadSureKeys(SureKeyFilter filter, SureKeySort sort,
                                                                    int skip, int limit)
```

This method is used to query for a list of sure keys as defined on the server. Filter is used to describe the records to search for. Sort defines the sort order of the values returned. Skip and limit are used to define the subset of objects that match the filter and sort that will be returned.

```
IList<ApiSureKey> ReadSureKeys (IEnumerable<SureKeyFilter> filters,
                                  IEnumerable<SureKeySort> sorts, int skip, int limit)
```

This method is used to query for a list of Sure Keys using multiple filters and sorts. The filters will be combined to produce an "AND" style query. The sorts will be applied in the list order. Skip and limit are used to define the subset of objects that match the filter and sort that will be returned.

Read Sure Keys Count

```
int ReadSureKeysCount(SureKeyFilter filter)
```

This method will return the number of records that match the filter.

```
int ReadSureKeysCount(IEnumerable<SureKeyFilter> filters)
```

This method will return the number of records that match the filters.

Create Sure Key

```
int CreateSureKey(int pkKeyset, string customerName, int violationLimit,
                 DateTime? trialExpiresDate, IEnumerable<ApiSureKeyProperty> properties)
```

This method will create a new Sure Key and will return the primary key of the newly created Sure Key. After the key has been created, call the ReadSureKey method to retrieve the readable 30 character key value.

Update Sure Key Customer Name

```
void UpdateSureKeyCustomerName(int pkSureKey, string customerName)
```

This method will change the customer name on a sure key.

Update Sure Key Violation Limit

```
void UpdateSureKeyViolationLimit(int pkSureKey, int violationLimit)
```

This method will change the violation limit on a sure key.

Update Sure Key Trial Expires Date

```
void UpdateSureKeyTrialExpiresDate(int pkSureKey, DateTime? trialExpiresDate)
```

This method will change the trial expires date on a sure key.

Update Sure Key Is Revoked

```
void UpdateSureKeyIsRevoked(int pkSureKey, bool isRevoked)
```

This method can be sued to revoke a sure key, or it can be used to remove the revoked setting on a sure key.

Delete Sure Key

```
void DeleteSureKey(int pkSureKey)
```

This method will delete a sure key and all records related to the sure key. Once the key has been deleted, it cannot be restored or recreated.

Create Sure Key Property

```
int CreateSureKeyProperty(int pkSureKey, int pkPropertyDefinition, string value)
```

This method is used to create a custom property value on a sure key. The value for pkPropertyDefinition is the primary key for the property definition.

Update Sure Key Property

```
void UpdateSureKeyProperty(int pkSureKey, int pkSureKeyProperty, string value)
```

This method is used to change the value on a sure key property.

Delete Sure Key Property

```
void DeleteSureKeyProperty(int pkSureKey, int pkSureKeyProperty)
```

This method is used to remove a property from a sure key.

API Errors and Exceptions

Any time an error occurs during an API method call on the server, an InstallKeyException will be thrown on the client side. Your calling code should handle these exceptions as you need.

All .Net exceptions have a Message property that is used to contain descriptive details about the error condition. However, this detail can become a security risk when someone attempts to glean system information from the errors by attacking the web service directly. Therefore, the message contents for API exceptions are intentionally ambiguous and contain no details.

However, during your testing and debugging you may want to see the details for the error messages. To see these, edit the web.config and change the ApiReturnErrorMessages setting to true. This will allow you to see the details in the Message property. After your development is complete, change the ApiReturnErrorMessages setting back to false.

14. Key Usage Invalidation

Key usage invalidation is the opposite of key validation. This process invalidates the local surety and removes the unique usage records from the server.

Invalidation Use Case

The typical use case is where a client validates a key, and sometime later wants to move the usage of the key to a different environment that will have different system identifiers. In this scenario, the key must be invalidated in the old environment, and the corresponding usage and identifier values must be removed from the InstallKey server, before the key can be validated on the new environment.

Invalidation of a Key Usage

The code below demonstrates how to invalidate a key usage from your client side code.

```
public bool InvalidateUsage() {
   ClientSideValidator validator;
   EnumInvalidateUsageResult result;

   // create an instance of the ClientSideValidator object
   validator = new ClientSideValidator(_keysetName, _password, _factor, _validationCode);
   validator.SetServiceAddress(_validationUrl);
   // check to see if the current local surety exists and is valid
   if (validator.IsValidated == false) {
       return(false); // there is no local surety to invalidate
   }
   // invalidate - remove the usage from the server and clear the local surety
   result = validator.InvalidateUsageAgainstServer();
   // check the result
   if (result != EnumInvalidateUsageResult.OK) {
       return (false);
   }
   // ok – usage successfully invalidated
   return (true);
```

Invalidate Code Sample

In the code above an instance of the ClientSideValidator object is created, the Service address is set, and it confirms that the current local surety is valid. Then the InvalidateUsageAgainstServer method is called to remove the usage and effectively decrement the usage count.

The InvalidateUsageAgainstServer method encapsulates the entire process and performs the following actions, in this order.

1. Retrieves the current local system identifiers.
2. Retrieves the local surety from SuretyStorage.
3. Confirms that the current identifiers match the identifiers in the local surety.
4. Invalidates the local surety.
5. Creates a surety object containing the sure key and the current system identifiers
6. Sends the surety object to the server via the Validator.svc web service.
7. The web service retrieves the usages and identifiers for the sure key from the database.
8. The web service compares the identifiers in the surety with the identifiers from the database. The corresponding usage record is found.
9. The key usage record is deleted from the database. This also deletes the related child Validation and Identifier records from the database.

10. The web service responds back with an OK status.

11. The InvalidateUsageAgainstServer method returns OK.

The end result is that the local surety is invalidated and the corresponding usage on the server has been removed and the usage count is decremented.

Result Codes and Error Handling

In the sample code snippet above, the returned result is compared against EnumInvalidateUsageResult.Ok to indicate a Boolean result. This is acceptable in a simple scenario; however, you might want to use the enumeration result to provide additional feedback to the user or other handling specific to your application.

Below is the description of the invalidate result codes, what they are indicative of, and why they might occur.

EnumInvalidateUsageResult.Ok
This value indicates that the usage invalidation against the server was successful. The usage and unique identifiers were successfully removed from the server, the usage count was decremented, and the local surety was also invalidated.

EnumInvalidateUsageResult.InvalidKey
This value indicates that the corresponding sure key that was used to create the local surety was not found on the server and the key record does not exist on the server. Typically this happens because the key was deleted on the server.

The locally stored surety will still be valid if the key is invalid. Depending on how you want to handle this condition, you may or may not want to call the InvalidateLocalSurtey method to remove the local surety.

EnumInvalidateUsageResult.NotFoundOnServer
This value indicates that a unique usage that matches the current identifiers was not found on the server. This condition could happen if the user invalidates the usage, restores the surety file and then invalidates the usage again.

The locally stored surety will still be valid if the usage was not found on the server. It is recommended that you call the InvalidateLocalSurtey method to remove the local surety, when this condition occurs.

Risks of Usage Invalidation

The risk to usage invalidation is the ease to which a user can circumvent the usage validation counts to use a key for more usages than allowed. A knowledgeable user may figure out that they can easily validate against the server, backup the surety, invalidate against the server and restore the surety. The user may or may or may not intentionally do this; the user might do this unintentionally by restoring a personal backup. The net effect of this is that the user will have a valid local surety, without a corresponding usage on the server.

If you utilize usage invalidation, you should consider the business implications and use the methodology that best fits your business needs and requirements. Consider one or more of these methods to mitigate the opportunity for a user to violate the key validation process by invalidating the key.

Do Nothing
This really is not a mitigating factor. In many scenarios this will be a perfectly acceptable risk, as this is the reason that you want to implement usage invalidation! You should consider how often will this really occur? How likely is it for your typical user to violate the key usage limitation by inadvertently restoring a backup of the surety? If this is not likely, and you trust that most users will be honest, doing nothing will be ok.

Surety Storage Location

You can implement a custom surety storage which will store the surety in a different location, making it less likely that the user will be able to restore a backup of the surety. This does not completely mitigate the chance for misuse, but it may make it more unlikely to occur.

Revalidate Periodically

Revalidating a key is the only way to truly mitigate the possibility for a user to misuse the key invalidation. You can revalidate the key against the server periodically, by calling the ValidateAgainstServer method, to ensure that the local surety has a corresponding usage on the server. This may already be happening anyway, depending upon how you have implemented InstallKey in you distribution.

The local surety contains the key that the user used to validate the first time, and this call to ValidateAgainstServer can be done transparently, perhaps whenever the application is started, or every few days to revalidate and refresh the key on the server. The downside to this is that the application must always have a connection to the internet; otherwise the lack of such a connection might frustrate your honest user. The more frequent you require your user to re-validate, the more likely the validation may annoy or make the user feel like the validation is "getting in the way."

Do Not Invalidate Usages

Another mitigating factor is to just not implement the invalidation of key usages in your application. By not invalidating the usage, you will force the user to contact you when they need to exceed their current violation limit. At this point you can increment the usage limit for the key on the server and the user will then be allowed to validate the key for another usage. (It's up to you to define you own human processes for how to handle this.)

15. Custom Identifiers and Matching

By default, InstallKey uses the Hard Disk, Bios, Network Adapter and the Processor ID as the source of the unique identifier values. For some use cases, this is more than enough; while in others, you may want to add additional identifiers. Or in other cases, you may want to replace all the default identifiers with identifiers of your own.

The default matching algorithms compare the current identifier values against the identifier values stored in the surety or against the identifier values stored in the database to ensure that a minimum number of matches exist. When calling the IsValidated property, the current identifiers are queried from the machine hardware and are compared against the identifiers stored in the surety. When calling the ValidateAgainstServer method the current identifiers are queried from the machine hardware and are sent to the server where they are compared against the identifiers stored in the database from prior key validations.

InstallKey provides interfaces that when implemented, allow you to modify or replace the default identifiers and matching algorithms. This provides unlimited possibilities for implementing custom identifiers and matching algorithms. For an example of how to implement custom identifiers, see the Web Application Sample.

The IIdentifierFinder Interface

The IIdentifierFinder interface is used to provide one or more of a single type of custom identifiers to the IdentifierFinders collection of the ClientSideValidator object. These identifier values are stored in the surety and validated against those values already on the server.

To provide your own identifiers, you will need to add one or more classes that implement the IIdentifierFinder interface to the IdentifierFinders collection of the ClientSideValidator object. Each IdentifierFinder implementation returns zero or more identifier values. Multiple, custom IIdentifierFinder types can be added to the IdentifierFinders collection. Also, any or all of the default identifiers can be removed.

In the examples that follow, InstallKey is being used to validate against only the machine name. This would result in a unique key usage for each machine where a key is validated that is based only on the machine name. To do this, the default identifiers will be removed and the custom machine name identifier finder will be added. Please notice that this is just an example on how to modify and use the identifiers and matching algorithms. The machine name may not necessarily a good identifier as it can easily be changed by the user.

The first step is to create a MachineNameFinder object that implements the IIdentifierFinder interface.

```
public class MachineNameFinder : IIdentifierFinder{

   public IList<InstallIdentifier> GetIdentifiers() {
       List<InstallIdentifier> list;
       InstallIdentifier identifier;

       list = new List<InstallIdentifier>();
       identifier = new InstallIdentifier(
                     "MachineName", Environment.MachineName);
       list.Add(identifier);
       return (list);
   }

   public string Name {
       get { return ("MachineNameFinder"); }
   }
}
```

An IIdentifierFinder Implementation

Next the default identifiers must be removed and replaced with the custom MachineNameFinder. The code snippet below shows how to replace the default identifier finders with the custom MachineNameFinder.

```
   string _password = "…";
   string _keysetFactor = "<keyset><Name>MyKeyset</Name>… …</keyset>";

   ClientSideValidator cl;
   IIdentifierFinder finder;

   cl = new ClientSideValidator(_password, _keysetFactor);
   cl.IdentifierFinders.Clear();
   finder = new MachineNameFinder();
   cl.IdentifierFinders.Add(finder);
```

Adding the Custom Identifier Finder

Now when the IsValidated property or the ValidateAgainstServer method is called, InstallKey will use the custom identifiers instead of the default identifiers.

Identifier Matching

Identifier matching is defined in the web application on the keyset by setting the minimum identifier matches when the Keyset is created. This matching compares the set of hashed values against those values stored in the database or the hashed values stored in the local surety. This occurs both on the server and on the client.

On the server, when a key is validated, the set of identifier hashed values are compared against the hashed values that were stored in the database on prior validations.

If the number of matches is less than the min number allowed for the keyset, then the server assumes that this is a new validation of the key. If the violation limit has not been met, the usage count is incremented and the new set of identifiers is stored in the database.

If the number of matches is greater than or equal to the min number allowed for the keyset, the server assumes that this is a revalidation of the key and does not increment the usage count.

On the client, when the local surety is validated, the matching is used to confirm that the application has not been moved or modified. The set of identifier hashed values from the surety are compared against the hashed values from the current environment, as provided by the Identifier Finders. If the number of matches is greater than or equal to the min number allowed, then the IsValidated property will return true. If the min number is less, then the IsValidated property will return false.

Custom Identifier Matching

Your business requirements for matching may be more than a simple count. For this reason, the ability to create your own custom identifier matching logic on both the client and server will be exposed as a public interface where you can add your own code.

Coming Soon! - This feature was excluded from the initial release of InstallKey version 5. It will be added to InstallKey in a near future release.

16. Web.config Settings

InstallKey is configured by default to provide secure error messaging and the best performance, and should not need to be modified. However, depending upon your implementation and requirements, you may want to change these settings. The following list describes the AppSettings values in the web.config file that you can change. Keep in mind that if you change the default settings, you do so at your own risk as described below.

ApiReturnErrorMessages

The ApiReturnErrorMessages can be set to true or false. The default is false.

Warning – setting this value to true in a production environment may allow someone who attacks the api service to glean system information. For this reason, this value should always be false in a production environment.

This setting effects the functioning of the error messages in the api calls. If any errors or exceptions occur on the server during an API call, the client side exception will not contain any detailed error information. To force the server to include the details, set this value to true. This may be helpful for debugging in your test or development environments.

ApiKeysetPropertiesLimit

The ApiKeysetPropertiesLimit can be set to any positive integer value. The default is 0.

Warning – changing this value will decrease the performance of the ReadKeysets api method call by adding the additional overhead of retrieving, populating and returning the additional content from the database.

This setting affects the functioning of the ReadKeysets() api call. By default the collection of ApiKeyset objects that are returned will not contain any Property Definitions. This is done to allow for the best performance.

However, your custom code might want to be able to see the property definitions on the returned ApiKeyset objects. To return the property definitions, change this value at the risk of affecting performance. The larger this value is the greater the risk that the API call will take longer to return. If you modify this value, it is recommended that you do not set this value over 50.

ApiSureKeyPropertiesLimit

The ApiSureKeyPropertiesLimit can be set to any positive integer value. The default is 0.

Warning – changing this value will decrease the performance of the ReadSureKeys api method call by adding the additional overhead of retrieving, populating and returning the additional content from the database.

This setting affects the functioning of the ReadSureKeys() api call. By default the collection of ApiSureKey objects that are returned will not contain any Properties, even when there are properties. This is done to allow for the best performance.

However, your custom code might want to be able to see the property definitions on the returned ApiSureKey objects. To return the properties, change this value at the risk of affecting performance. The larger this value is the greater the risk that the API call will take longer to return. If you modify this value, it is recommended that you do not set this value over 50.

17. Hosted Deployment

InstallKey is designed such that the InstallKey web application can be deployed to any third party hosting provider. The web application utilizes basic ASP.Net functionality, WCF and SQL Server database access logic that all hosting providers can be expected to support.

Hosted deployment is typically a manual process, and may require the source code edition of InstallKey and some additional skill. However, if you are comfortable creating IIS virtual folders, configuring ASP.Net web applications and managing SQL server databases with your favorite hosting provider, you should not have any problems. Just follow these steps and guidelines.

The process of deploying the InstallKey web application will vary depending on your hosting provider's toolset. However, the same methodology that you would use to create and deploy other ASP.Net web applications is the same process that you will use for deploying InstallKey. Please consult your hosting provider's documentation for instructions on how to create virtual folders, web applications, upload content files, and manage SQL Server databases. Once you are comfortable with your hosting provider's processes and tools, you will be able to deploy InstallKey.

The common steps to deploy the web application are as follows.

1. Create the InstallKey SQL Server database in your hosting provider's environment using your hosting provider's tools and methods. Execute the InstallKeyDatabase.sql script to build the tables, views and other schema. Or if your provider allows, you can backup and restore an existing copy of the database created locally.
2. Using your hosting provider's tools, create the SQL Server credentials that the InstallKey web application will use to read and write to/from the database. Give these credentials access to the database.
3. Using your hosting provider's tools, create the web application and/or virtual folder for the InstallKey web application. Copy the application files and folders.
4. Edit the InstallKey web.config file. Set the connection string to the proper SQL Server database and credentials.
5. If all is set up and configured properly, you should now be able to open a browser window to your installation of InstallKey. You should be able to navigate to the log in page and log into the web application.
6. From the Validation page, apply your key to validate your installation to your hosting provider's server so that you can create an unlimited number of keysets and sure keys.

Test your deployment. If all is correctly installed, you should be able to view the Admin web pages, create keysets and keys, and validate keys. Once installed, you can validate your key to remove the demo restrictions.

Validation note – InstallKey uses InstallKey validation and the server hardware identifiers to uniquely identify the installation, so this may have an effect when validating your key for InstallKey. In most cases, there should not be an issue. However, some web hosting providers may lock down and prevent access to WMI calls, thus preventing your deployment from validating your InstallKey surety at runtime and from the key validation page. Or, your hosting provider may dynamically move your web site to another server, which may result in a change to the identifiers. If this were to happen, it will only prevent you from creating new keys, although this will not affect the validation of existing keys. If this is an issue, it can be resolved if you purchase and rebuild the InstallKey web application source code to remove the InstallKey activation.

18. Rebuilding the Source Code

If you purchased the source code for InstallKey, you will be able to rebuild the web application and client side dlls using Visual Studio 2010 or newer.

Keep in mind that you are still bound by the LomaCons InstallKey license agreement when you rebuild the source code. If you rebuild the LomaCons.InstallKey.Common.dll and LomaCons.InstallKey.ClientSide.dll you must obfuscate these dlls to redistribute them with your application. In addition, you cannot redistribute or resell any of the server side code, database schema or database logic.

To rebuild the web application and client side dlls, unzip the InstallKeySource.zip file. Open the InstallKey.sln file with Visual Studio, and rebuild the solution.

No source code changes are needed to remove the InstallKey license validation. Once you have rebuilt the source code, the resultant web application does not require key validation against the LomaCons key validation server to remove the demo limitations. The rebuilt application will be able to create an unlimited number of keysets and keys.

19. Upgrading from InstallKey 4.2 or 3.4

Starting with version 5.0, it was not possible to maintain 100% backwards compatibility with prior releases of InstallKey. The differences in .Net web services, and a desire to modernize InstallKey, have resulted in a decision to make a clean break from prior versions.

The most significant change is that the .asmx web services that have been used for validating keys since version 1.0 of InstallKey have been replaced with modern .Net 4.0 based WCF services. This means client code based upon the InstallKey 4.2 (or earlier) client side dlls cannot validate against the InstallKey 5 server. The recommended way to maintain compatibility is to have two validation websites.

For example, you will keep your current InstallKey 4.2 (or 3.4) web application installation running version 4.2, which can be used to validate prior releases of your software, and a new web application running version 5 for validating new releases of your software that contain the new client side code.

Although not fully backward compatible with prior versions, there are two upgrade paths that will allow you to carry your keysets, keys, identifiers and usage counts from Version 3.4 and 4.2 forward into version 5. This allows you to manage and validate the same 30 character keys in both versions of InstallKey.

Full Migration

One upgrade method is to do a full migration of the InstallKey database content. To do so, you will need to copy the database values from the prior version of InstallKey to the database used by the new version.

Copying the database content from prior versions to the new version can be done using your database tools or an SSIS package. Map the values from the prior database to the new database following tables below. Creation of the code to migrate the content is not provided but can be easily created following the guidelines below.

The primary key values in the version 5 database will be created automatically by the SQL Server identity insert and must be mapped to the corresponding foreign key columns in the new tables. The primary key and foreign key values are arbitrary and do not need to be the same as the values in the version 4.2/3.4 database.

Version 5 Column	Version 4.2/3.4 Column	Notes
Keyset.Name	Keyset.Name	
Keyset.Password	Keyset.Password	
Keyset.Factor	Keyset.Factor	
Keyset.MinMatches	--	Set to 3
SureKey.FKKeyset	--	Must match the corresponding Keyset.PKKeyset value in the Version 5 database.
SureKey.GuidValue	InstallKey.InstallGuid	
SureKey.KeyValue	InstallKey.InstallKey	
SureKey.ViolationLimit	InstallKey.ViolationLimit	
SureKey.IsRevoked	InstallKey.Revoked	
SureKey.CustomerName	InstallKey.CustomerName	

Version 5 Column	Version 4.2/3.4 Column	Notes
SureKey.TrialExpires	InstallKey.IsTrial InstallKey.TrialExpires	If InstallKey.IsTrial is false, set SureKey.TrialExpires to NULL, otherwise set SureKey.TrialExpires to InstallKey.TrialExpires
KeyUsage.FKSureKey	--	Must match the corresponding SureKey.PKSureKey value in the Version 5 database.
UsageValidation.FKKeyUsage	--	Must match the corresponding KeyUsage.PKKeyUsage value in the Version 5 database.
UsageValidation.ValidationDateUtc	KeyUsage.CreatedDate	
UsageValidation.Content	--	Populate with an empty string N''
Identifier.FKKeyUsage	--	Must match the corresponding KeyUsage.PKKeyUsage value in the Version 5 database.
Identifier.Name	Identifier.Name	
Identifier.Value	Identifier.Value	

Individual Key Import

Individual keys and keysets can be imported from prior versions of InstallKey using the web pages provided in the InstallKey Web application. This feature will automatically be visible in the web user interface if a prior InstallKey version 3.x or 4.x database is defined and found.

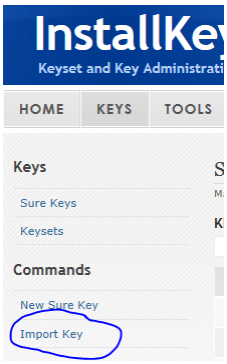
To define the location of the previous database, create a connection string for the InstallKeyPrev database in the web.config. This database connection string can be defined by manually editing the web.config, or by creating a connection string in the IIS administration console.

The code block below shows the connection string section from the web.config file. Notice that both the connection to the InstallKey database and the previous database are defined. (The connection string itself is specific to your environment.)

```
<connectionStrings>
  <add name="InstallKey" connectionString="Data Source=… "/>
  <add name="InstallKeyPrev" connectionString="Data Source=… "/>
</connectionStrings>
```

InstallKey will attempt to connect to the InstallKeyPrev database only once using the connection string provided, and will store the result in the web application cache. Because the setting is cached, to get InstallKey to detect that database, it may be necessary to restart the web application or the application pool if you make changes to the connection string or database server.

If the database connection is defined correctly, and the database is a version 3.x or version 4.x database, the Import Key link will be visible on the SureKey Maintenance page.



Import Key Link

From the Import Key page, find the key that you want to import from the previous database. When imported, the key, its usage and identifier values are copied into the InstallKey 5 database. If the keyset does not already exist in the InstallKey 5 database, the keyset will be imported too.

Note: Keysets are unique, even when the keyset name is the same. Do not create the keyset in InstallKey 5. Rather you should allow InstallKey to import the keyset. When the first key of a keyset is imported, the keyset will be imported automatically with the same keyset factor and password as was defined in the previous version.

20. Future Enhancements

Many additional new features and customizations are planned for future releases. Any additional requests for new features or enhancements can be made by contacting LomaCons. See www.lomacons.com for information on how to contact LomaCons.

21. Development Methodology and Goals of Version 5

Version 5 was a major retooling and refactoring of InstallKey to benefit InstallKey developers and application integrators, targeting both current developers and new developers. The goals of the first release of version 5 were as follows. Many of these continue to be guiding principles for current a future releases of InstallKey.

Make InstallKey easy to use for those new to InstallKey Validation
We want to ensure that those who begin to use InstallKey can start working with it right away, with just a small learning curve. It should take no more than a few minutes for someone new to InstallKey to get the application installed and start working with it.

Update InstallKey to use modern Microsoft .Net framework tools and technologies
This included dropping the old .asmx web services in favor of modern WCF services. Also dropped was support for older versions of Windows and SQL Server.

Make the difficult features easy
One of the most difficult features to use from prior versions of InstallKey was the custom properties and the API. Version 5 makes adding custom properties easy, and does not require the software integrator to write any custom code or ascx user interface components which were difficult to develop and deploy.

Another difficult feature from prior version was the API. The new InstallKey API is integrated directly into the client side dll as simple to use method calls. This encapsulates the web service call so that the software integrator does not need to directly reference and manipulate complex web services.

Make installation, deployment and configuration easy
For this, a single web application that hosts the validation service and the key management website was chosen as the new deployment model. The multiple web applications, and odd configuration settings from prior versions is gone. InstallKey 5 is simple to understand and uses commonplace patterns and practices.

Realistically support an unlimited number of keys
The version 5 web application has been rewritten from the ground up to be easier to use, even when there are thousands or millions of keys. Prior to version 5, the out of the box key administration web application was lacking. The user interface was difficult to use for more than a few hundred keys, which meant than many integrators would have to write their own administration tool.

The new user interface is clean and professional, and contains new functionality that allows you to sort, filter and quickly locate and manage your keysets and keys. Hopefully there will be little need for anyone to have to write their own key administration user interface with this new version. (Although the ability to do so using the API will be carried forward, for those software integrators that wants to.)

Stay true to core values of less-is-more, easy and customizable
InstallKey 5 contains many of the same customizations that were available in prior releases. Where functionality has been initially omitted in version 5, we will be adding ways to implement these soon. The best part about this new version is that the break from prior versions will allow future changes to be implemented and released quicker.

Provide an upgrade path and backward compatibility as best as possible
The significant differences in .Net over the years, and a desire to modernize InstallKey have resulted in some breaking changes. Although not fully backward compatible with prior versions, there is an upgrade path that will allow you to carry your keysets and keys from Version 3.4 and 4.2 forward into version 5.

22. Product Support and Contact Information

All product support is provided by web and email.

Frequently Asked Questions

Frequently asked questions and FAQs can be found at http://www.lomacons.com.

Email Support

Support for InstallKey can be sent to support@lomacons.com. In most cases, you should receive a reply by the next business day or sooner.

Sales and Information

Product sales can be reached by email at support@lomacons.com.