

The Desaware Licensing System

End to end cryptographic
machine/server licensing for .NET.

Version 1.6

for Visual Studio .NET

Desaware, Inc.

Rev 1.6.1 (11/09)

Desaware, Inc. Software License

Please read this agreement. If you do not agree to the terms of this license, promptly return the product and all accompanying items to the place from which you obtained them.

This software is protected by United States copyright laws and international treaty provisions.

This program will be licensed to you for use only on a single computer. If you wish to install it on additional computers, you must purchase additional software licenses. You may (and should) make archival copies of the software for backup purposes.

You may not make copies of this software for other people. Companies or schools interested in multiple copy licenses or site licenses should contact Desaware, Inc. directly at (408) 404-4760.

You have a royalty-free right to incorporate any of the sample code provided into your own applications with the stipulation that you agree that Desaware, Inc. has no warranty, obligation or liability, real or implied, for its performance.

Licensing: The Desaware Licensing System uses the Desaware Licensing System Component. This framework provides for the transfer of licensing information from the system upon which the Desaware Licensing System is installed, to Desaware's Licensing Web Service. This in turn creates and activates a license key that allows you to use the Desaware Licensing System components and services on your system. The licensing information transferred is a one way cryptographic hash that does not include any personal information, or information that could be used to identify the originating system. If you perform online registration, the registration information will also be transferred.

File Descriptions: You may distribute the files Desaware.MachineLicense?.dll, Desaware.webcodeentry?.dll and Desaware.Dls.Interfaces?.Dll with your licensed applications. No other files may be redistributed (?? indicates the framework version).

Source Files: If you have purchased a source code license, the following applies: You may rebuild modified versions of the software provided subject to the restrictions listed. You may not use this source code to develop or distribute components and tools that provide functionality similar to all or part of the functionality provided by the Desaware Licensing System or any of its components except for use licensing your own applications. Modified assemblies and namespaces must be renamed – you may not use Desaware in the assembly name or any namespace. However Desaware's copyright notice must be prominently displayed in any location where your own copyright notice is present. Source code may not be published or distributed, and may be used or accessed only by the individuals and at the locations covered by the source code license. You may distribute modified versions of the Desaware.MachineLicense?.Dll or Desaware.WebCodeEntry?.DLL files only for use by your applications. Modified versions of other components may only be used by the individuals and at the locations covered by the source code license. Modified versions of the Desaware.MachineLicense?.Dll, Desaware.WebCodeEntry?.Dll and Desaware.LicenseServer?. files must be obfuscated before redistribution.

Limited Warranty

Desaware, Inc. warrants the physical CD and physical documentation enclosed herein to be free of defects in materials and workmanship for a period of sixty days from the date of purchase.

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective CD(s) or documentation and shall not include or extend to any claim for or right to recover any other damages, including but not limited to, loss of profit, data or use of the software, or special, incidental or consequential damages or other similar claims, even if Desaware, Inc. has been specifically advised of the possibility of such damages. In no event will Desaware, Inc.'s liability for any damages to you or any other person ever exceed the suggested list price or actual price paid for the license to use the software, regardless of any form of the claim.

DESAWARE, INC. SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Specifically, Desaware, Inc. makes no representation or warranty that the software is fit for any particular purpose and any implied warranty of merchantability is limited to the sixty-day duration of the Limited Warranty covering the physical CD and documentation only (not the software) and is otherwise expressly and specifically disclaimed.

This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

This License and Limited Warranty shall be construed, interpreted and governed by the laws of the State of California, and any action hereunder shall be brought only in California. If any provision is found void, invalid or unenforceable it will not affect the validity of the balance of this License and Limited Warranty, which shall remain valid and enforceable according to its terms.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of Commercial Computer Software - Restricted Rights at 48 CFR 52.227-19, as applicable. Contractor/Manufacturer is Desaware, Inc., 3510 Charter Park Drive, Suite 48, San Jose CA, 95136.

Microsoft is a registered trademark of Microsoft Corporation. Visual Basic, Visual Studio, Windows, Windows 95, Windows 98, Windows ME, Windows NT, Windows 2000, and Windows XP are trademarks of Microsoft Corporation. Desaware Licensing System, FiveMinuteSoftware, CAS/Tester, SpyWorks, NT Service Toolkit, StateCoder, VersionStamper, StorageTools, Event Log Toolkit, ActiveX Gallimaufry, Custom Control Factory, and SpyNotes #2, The Common Dialog Toolkit are trademarks of Desaware, Inc.

Information in this document is subject to change without notice and does not represent a commitment on the part of Desaware, Inc. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Desaware, Inc.

Copyright © 2003-2009 by Desaware, Inc. All rights reserved. Printed in the U.S.A.

Important Notice! Warning! Danger!

If you implement a scenario that requires activation in order for your software to install, and at some future date your licensing server becomes unavailable, that software will become uninstallable. It will not be possible to enable features which require activation. This system uses 128 bit encryption, and there are no backdoors or secret codes built in that will allow you to bypass this licensing scheme. Neither do we know of any flaws at this time that might allow you to do so.

We ourselves have no tools or technology that would allow recovery or regeneration of the private key necessary to re-enable licensing of software if the original license server database is lost.

We encourage you to choose the 'Friendly Security' scenario if you wish to allow your software to be installable/runable without server activation (even though it does not provide the same level of security).

We strongly recommend you take precautions to back-up your database.

Important Notice #2! Warning! Danger!

Any licensing scheme is vulnerable to reverse engineering – someone completely decompiling, modifying and then recompiling your software. Due to the nature of .NET software, it is potentially more vulnerable to this type of attack than traditional software. It is essential that you make use of obfuscation to reduce the chance of your assembly being decompiled in this way.

Version 1.1 and later of the licensing system includes an obfuscator, and additional binding test code that you can incorporate into your software to prevent this type of attack. Refer to the application note StrongName.pdf for an in-depth discussion of this topic.

Important Notice #3!

Samples and Debugging!

The number one technical support call we get is when the following exception occurs:

The MachineLicense component must itself have a valid license to Install software when running under a debugger. Be sure to copy a valid .DLSC file to the same directory as the Desaware.MachineLicense?.dll

In other words - the machine license component is itself licensed. You need to place the dls10client.dlsc file that was created during installation into the bin directory of your application in order to debug it. This is, of course, only required to debug your licensed application (during development). **This is also required to debug any of the sample programs!**

To run samples:

Most of the examples include a sample resource file Test.resx or Test15.resx in their application or App_GlobalResources directory. You **MUST** replace this with a resource file for an application on your own licensing server in order to successfully run or test these examples!

Table of Contents

QUICK INTRODUCTION	10
On Version and Framework Numbers	11
Framework version	11
Assembly versions	11
File versions	12
Previous Versions	12
WHAT'S NEW IN VERSION 1.3-1.6	13
Version 1.6 Update	13
General Features	13
MachineLicense	13
License Server	14
SET UP AND INSTALLATION	15
The Licensing Service	15
Selecting the Server	15
Selecting the Virtual Directory	16
Selecting the Database	17
Securing the Web Service	18
Entering Licensing Information	19
Uninstalling the License Server	20
Testing and Debugging the License Server	20
The License Manager and Development Components	22
THE LICENSE MANAGER	23
License Manager Security	23
Using the License Manager	23
Menu – Advanced - Set Server Connection	24
Menu – Application - Create New Application	25
Developer Information	26
Sign External DLSC	28
Menu - Keys - Create New Keys	29
Installation Keys	29
Menu – Keys - Find Key	31
Individual Key information	31
Advanced License Manager Options (Hosted Installations)	32

Advanced License Manager Options (Binding Code Generator)	34
LICENSING SCENARIOS AND SAMPLES	36
On Strong Names	36
Preparing to Use the Samples	37
High Security Scenario	37
Implementing the High Security Scenario	37
Creating a Licensed Application	38
Verifying if an Assembly is Licensed	38
Licensing the Assembly	39
Variations	41
Potential Vulnerabilities	41
More “Friendly” Scenarios	42
Implementing the Friendly Security Scenario	42
Creating a Licensed Application	42
Verifying if an Assembly is Licensed	43
Using the CodeEntryControl Control	44
Licensing the Assembly	45
Handling Deferred Licensing Results	46
Variations	48
Potential Vulnerabilities:	49
The InstallerLicenseTool Example	49
Licensing Components	49
Potential Vulnerabilities	53
Variations	53
Embedded Components	54
Using Custom Data	54
CLIENT COMPONENT REFERENCE	55
ClientLicense Properties	55
ClientLicense Methods	60
ClientLicense Events	66
SaveLicenseModes Enumeration	68
ValidationStatus Enumeration	70
InstallErrorResults Enumeration	70

The CodeEntryControl	72
CodeEntryControl Properties	72
CodeEntryControl Methods	72
CodeEntryControl Events	73
Additional Information	73
Code Access Security	73
How the Component is Licensed	73
Code Reuse Blocking	73
THE WEBCODEENTRY CONTROL	75
Using the Desaware.WebCodeEntry control	75
Desaware.WebCodeEntry control properties	75
Client Side Support	75
WEB SERVICE REFERENCE	77
Connecting to the Web Service	77
Web Service Methods	77
Management Web Service	77
Activator Web Service	82
EXTENDING THE LICENSING SYSTEM	83
System Identifiers	83
System Matching Algorithms	85
Installation Match Algorithm	85
Demo Match Algorithm	86
Defining a Custom System Match Algorithm	86
Creating Post Installation Actions	89
Adding data to license certificates	91
Additional ServerData Considerations	96
THE DESAWARE LICENSING SYSTEM: INTERVIEW WITH THE ARCHITECT	97
MORE FAQ'S	105
APPENDICES	107

Glossary	107
Configuring the Web Service web.config File	107
Database Schema and Contents	108
Application Table	108
InstallationCodes Table	108
UniqueInstalls Table	109
SystemIdentifiers Table	109
CustomData Table	109
Installation and Existing Certificates	109
Working with proxy servers	112
Version History - 1.2 Update	113
License Server	113
MachineLicense	113
Samples	114
Utilities	114
Single Application Edition	114

Quick Introduction

The Desaware Licensing System combines strong licensing security with ease of use. For a proper introduction, we strongly recommend you read “The Desaware Licensing System: Interview with the Architect”, which lays out the philosophy and architecture of the system.

For those who wish to use the system as quickly as possible, here is a very brief introduction.

For ease of understanding, we’ve divided the manual description of the system into three common scenarios (though each one can be implemented in a wide variety of ways).

High Security scenario	This scenario requires server activation via Internet or other connection. The server returns to the client a licensing certificate (DLSC file) that is digitally signed. The licensing certificate can only be used on that machine.
Friendly Security scenario	This scenario uses a temporary unsigned certificate for cases where an Internet connection does not exist or a licensing server is unreachable. The licensing component will attempt to contact a server at a later time to complete verification and the licensing process.
Component scenario	Allows licensing of components, with or without using the LicenseProvider architecture defined by .NET.

All of these scenarios support limited duration demonstration licensing.

Additional samples an applicatioin notes demonstrate additional scenarios including:

- Licensing of modules in an application
- Subscription/time based license expiration
- Concurrent (lease based) licensing
- Licensing based on user, domain/IP address, or other criteria

On Version and Framework Numbers

There are three different version numbers to consider when working with the licensing system: the .NET framework version, the Assembly version and the File version.

Framework version

It is always best to run .NET components and applications on the same framework version they were built on. Fortunately, the .NET framework supports side-by-side installation, meaning a system can support multiple versions of the framework at once.

We've adopted the convention of including the target framework version for each component in the component name. For example: the Desaware.MachineLicense20.dll component targets the .NET 2.0-3.5 framework.

Some framework versions are distinct – the 1.1, 2.0 and 4.0 frameworks can exist on the same system and you should create separate applications for each one.

The 3.0 and 3.5 frameworks are extensions to the core 2.0 framework. This means that any assembly written for 2.0 will run on 3.0 and 3.5. However, assemblies that target 3.0 or 3.5 may not run on 2.0 because it may lack required features.

Assembly versions

The assembly version is part of the “strong name” for an assembly. This means that as long as the assembly version remains the same, you can drop in an update without having to rebuild your application.

We have shipped three different assembly versions of the Desaware.MachineLicense20.dll component to date:

Assembly version 1.2 – Initial release on the .NET 2.0 framework.

Assembly version 1.3 – Vista support

Assembly version 1.5 – Numerous feature enhancements (see version history)

Assembly version 1.6 – This release

All versions of the license server for the .NET 2.0 framework will support all client components.

When upgrading assembly versions, you must rebuild your existing applications to reference the newer component.

As an alternative to rebuilding your application, you can add the following to the <configuration> section of your application's configuration file as long as the framework version is the same. **Note – you must add this AFTER the <configSections> tag, which must always be the first one in the configuration file.**

```
<runtime>
  <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
    <dependentAssembly>
      <assemblyIdentity name="Desaware.MachineLicense20"
        publicKeyToken="c8956dcc7a600871"
        culture="neutral" />
      <bindingRedirect oldVersion="1.1.0.0"
        newVersion="1.5.0.0"/>
    </dependentAssembly>
  </assemblyBinding>
</runtime>
```

File versions

Each assembly version may be upgraded in cases where we are confident that the newer component is completely backward compatible with the prior version. In these cases we increment the file version. You can use the file properties viewer to see the file and assembly versions of components.

Previous Versions

We are currently pursuing active development on the .NET 2.0-3.5 frameworks. .NET 4.0 will be supported on release.

The last original .NET client component is assembly version 1.1.0.0, file version 1.1.0.0.

The last .NET 1.1 client component is assembly version 1.1.0.0, file version 1.3.0.0.

What's new in version 1.3-1.6

Version 1.6 Update

The version 1.6 update incorporates a wide variety of improvements to the licensing system, while maintaining backward compatibility with exiting systems.

General Features

This update includes both Licensing System Application notes including a concurrent licensing example with server, and a subscription application with management features.

MachineLicense

The following versions of the MachineLicense component are included in this release:

Desaware.MachineLicense20.dll version 1.5.3.0 (latest 1.5 assembly version)

Desaware.MachineLicense35.dll version 1.6.0.0

The feature updates for each version are as follows:

Version 1.6

- FIPS compliance
- Native WPF code entry control
- Application Note: Advanced subscriptions and extending demo certificates
- Application Note: Concurrent Licensing (floating licenses)

Version 1.5

- Flexible license certificate storage including Isolated Storage and customized storage.
- Web code entry control for use on web sites
- Added paste feature to CodeEntryControl.
- Ability to remove default identifiers.
- New constructor allows embedding license resource in any assembly.
- New properties: InternalLicensePath, Certificate, TimeServerSettings.
- New method GetActivationServerTime
- Additional functional and performance improvements.

Version 1.3

Vista compatibility

License Server

This update includes the version 1.6 LicenseServer application.

There is no reason for you to update your existing license server if it is working correctly.

Other than the update to 1.3 (which offered Vista compatibility) all updates through version 1.6 have been minor changes to improve support on different server configurations.

The version 1.6 server is licensed for use by URL/Application on Cloud/fallback servers and works correctly in that environment.

Set Up and Installation

The Desaware Licensing System includes three major components:

Licensing Service	This is a web service that runs on any machine that hosts IIS.
License Manager	This is a Windows application that is used to define applications, create installation codes, and perform other tasks necessary to use the Desaware Licensing Service.
MachineLicense Component	This is the redistributable component that must be included with each piece of software that you are licensing.

The Desaware Licensing System is partitioned into two packages, each of which has its own installation and license key.

- The Licensing Service
- The License Manager and Development Components.
This includes the documentation and the sample programs as well.

There are a number of choices involved during the installation process that will need to be considered. Please read the following installation guide carefully.

The Licensing Service

The Licensing Service is a web service that is accessed by licensed components during the installation process. It can be installed on any ASP.Net compatible server. The server will access a database that you define. During installation you have the opportunity of specifying any accessible SQL server database, an Access database (if the OleDB JET drivers are installed), or you can specify an OleDB connection string to the database of your choice.

- The licensing server can support an unlimited number of applications in a single database (the single application edition only supports a single application).
- The Licensing Service is installed using the Desaware.LicenseServer?.msi installation package.

The License Server can also be used on hosted installations (where an installation file may not be run) if certain requirements are met. Refer to the section “Advanced License Manager Options (Hosted Installs)” for details.

Selecting the Server

Each server license you purchase entitles you to install the LicenseServer on a single system. In order to deploy the system, you obviously need at least one LicenseServer running on a system accessible to your clients (either on the Internet, or on an intranet for internal applications).

Your developers can use the same LicenseServer. You may also wish to deploy a separate license server just for development use. In either case, all your developers can share a single license server – you do not need one on each developer machine.

Selecting the Virtual Directory

The first choice you have to make is the name of the virtual directory in which to install the Licensing Server.

This decision is important, because it determines the URL that will be used to access the Licensing Server. The default is LicenseServer under the default web on the server. You can change the name from LicenseServer to something more cryptic if you wish. You can also select the application pool for the server.

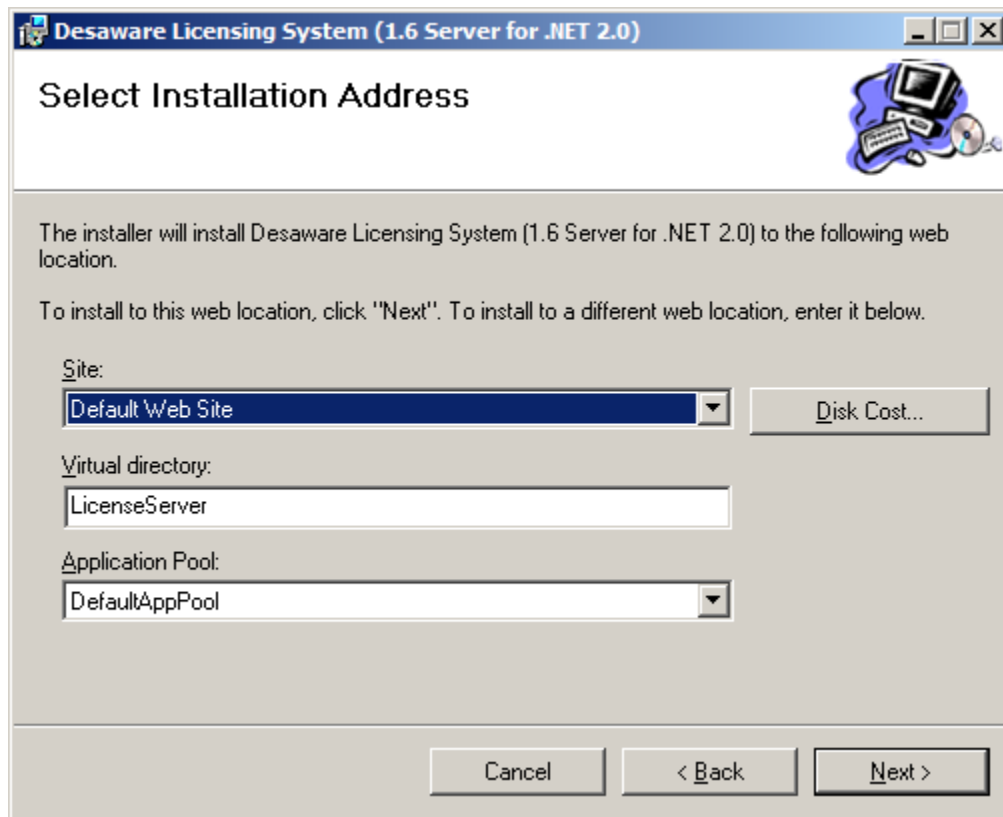


Figure 1
Server Installation Dialog Box

Remember the URL used to access your licensing server. You'll need it to connect to the server later. You can change the port number of the server or specify host headers or perform other configuration tasks using the standard IIS management console.

Selecting the Database

After several progress forms, you will come to the License Server configuration form.

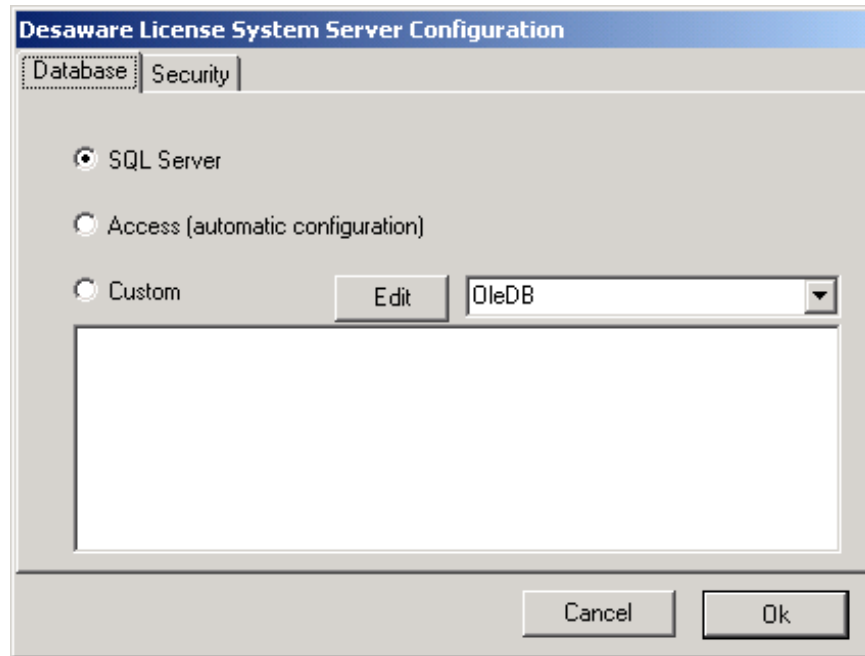


Figure 2
License Configuration Form

The SQL Server option will be available if there are one or more instances of SQL server accessible to your system (in this example, there is a single instance running). If there were more than one you would be able to choose among them). Instances of the MSDE or SQL Server express database will be included here as well.

If you wish to use SQL Server, it is up to you to install it before installing the Desaware License Server. The administrator account that is installing the licensing server must have permission and access to the instance of SQL server in order for the automatic installation to succeed.

If the installation program is unable to complete the automatic installation, you will see a prompt warning you that you must configure the database manually. You can view diagnostic information explaining why the automatic configuration failed in your application event log.

You can also choose to use an Access database. As each client only connects with the licensing service once to obtain a certificate, the service imposes a light load on the system, and does not generally require a high throughput database. Access should be adequate for all but the most high volume sites.

You may also specify a custom database connection string. The Edit button brings up the OleDB connection editor that allows you to connect to a variety of databases. However, note that the system has only been tested with SQL Server and Access, so we can't

guarantee operation if you do something “creative” like trying to use OleDb to connect to an Excel spreadsheet.

If you specify a custom connection string, you should also choose the provider type: OleDb, SQL or ODBC.

You can create a new database or use an existing database. If you use an existing database, the Licensing Server will create the following tables:

- Application
- CustomData
- InstallationCodes
- SystemIdentifiers
- UniqueInstalls

If your database is secured, you **MUST** use a custom connection string that includes the necessary user information and password to access the database.

Refer to the Database Schema documentation for more details on the structure of the database.

Securing the Web Service

The Security tab is used to secure the management features of the database.

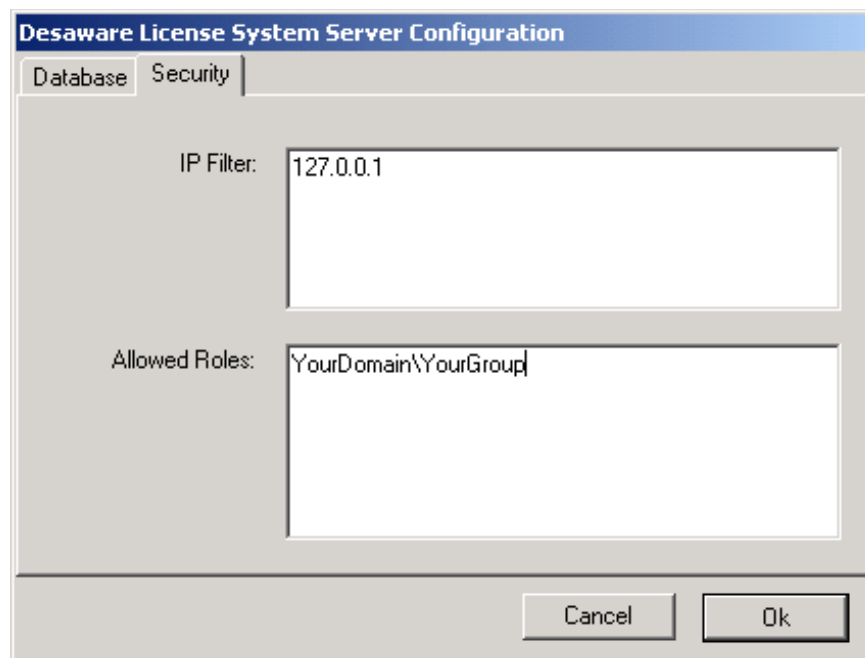


Figure 3
Security Dialog Box

The Licensing web service has two entry points, Activator.asmx and Management.asmx. The Activator.asmx entry point is used to register licenses and return licensing certificates and is therefore open. Security to this point is provided by the fact that it only has a single function that accepts data in a very tightly controlled and encrypted format.

The Management.asmx entry point is used to manage the licensing system and should be available only to those systems in your network that are allowed to perform these operations.

Two types of built-in security are provided (plus, you can use integrated Windows authentication as well).

1) You can specify IP Filters in the form.

a.b.c.d – to specify a single IP address

or

a.b.c.d-e.f.g.h – to specify a range of IP addresses

2) You can place each permitted IP or range on a separate line.

You can also specify a role in the form *YourDomain\YourGroup*, where *YourDomain* is the name of the domain or computer, and *YourGroup* is the name of a group or account.

Access to the management features are allowed if the requesting computer is in the valid IP range OR the role is allowed (either is acceptable).

If you use roles, the Management.asmx entry point to the service will be configured to prohibit anonymous access and only use NTLM authentication. This means that once you specify a single role, access will only be possible to clients that connect using NTLM and can authenticate on the server. However, the service will accept any authenticated client even if the role does not match, as long as it comes from a valid IP.

You must have NTLM authentication enabled on your server for role authentication to function!

You can use IIS Manager to also enable Basic authentication if you wish.

The default configuration allows access to management features from the server only (account 127.0.0.1 is the local system).

Refer to the Appendix “Configuring the Web Service web.config file for more information”.

3) If you configure the Management.asmx entry point to use NTLM authentication, the License Manager application will, if necessary, prompt the user for a user ID and password to use when accessing the web service (much as a web browser will prompt for a user ID and password when accessing a secured web site).

Entering Licensing Information

The next window is where you enter the installation code provided by Desaware. You can also register online as part of the activation process, but that part is optional. The setup program will automatically detect whether you are installing the full version or single application version of the licensing system based on the installation code.

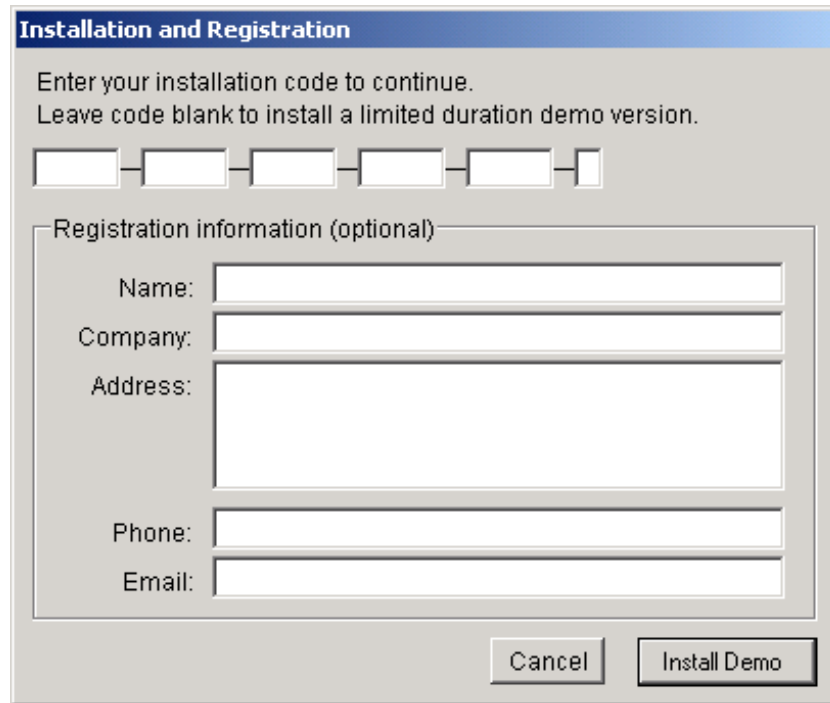
The image shows a Windows-style dialog box titled "Installation and Registration". It has a blue header bar. Below the title, there is instructional text: "Enter your installation code to continue. Leave code blank to install a limited duration demo version." Below this text is a row of six small, empty rectangular input boxes. Underneath the boxes is a section titled "Registration information (optional)" which is enclosed in a thin border. This section contains several labels and corresponding input fields: "Name:" followed by a single-line text box, "Company:" followed by a single-line text box, "Address:" followed by a larger multi-line text box, "Phone:" followed by a single-line text box, and "Email:" followed by a single-line text box. At the bottom right of the dialog box, there are two buttons: "Cancel" and "Install Demo".

Figure 4
Licensing System Installation and Registration

The sample code for the installer class we use is included with the Desaware Licensing System (part of the development system installation), so you can use this approach with your own customers as well.

Uninstalling the License Server

Uninstalling is automatic except in one situation. If the License Server originally created a database, you will be prompted to remove the database if you wish.

WARNING! If you remove the database, the private keys and information required to install any licensed applications will be permanently lost. Be sure you have backed up your database before removing it.

The uninstall program will prompt you twice for confirmation, to minimize the risk of accidental deletion.

Testing and Debugging the License Server

The installation program does an excellent job of installing and configuring the licensing server, but it is possible that configuration errors will occur – for example: if security settings are incorrect or if the database connection string is incorrect. In addition, problems often occur

during hosted installations (described in the section “Advanced License Manager Options” later in this document).

If you run into difficulty, please download the latest version of our Server Configuration Guide available at

<http://www.desaware.com/support/documents/files/LicensingServerConfiguration.pdf>

This guide is continuously updated with the latest information based on customer feedback from various servers and hosts.

By default, all errors that occur on the licensing server are hidden. In order to determine which problems are occurring, you can temporarily expose the service features so that you can test them using a web browser.

To do so, make the following changes to the web.config file on the license service:

1. Comment out the wsdlHelpGenerator tag as follows:

```
<!--  
  <webServices>  
    <wsdlHelpGenerator href="helppage.htm" />  
  </webServices>  
-->
```

This will allow you to directly access the web service functions from the browser.

2. Turn off custom errors by changing the customErrors mode from RemoteOnly (the default) to Off as follows:

```
<!-- CUSTOM ERROR MESSAGES  
  Set customErrors mode="On" or "RemoteOnly" to enable  
  custom error messages, "Off" to disable.  
  Add <error> tags for each of the errors you want to  
  handle.  
-->  
<customErrors mode="Off" />
```

You can test the access security settings of your license server by accessing the server on a browser using the following URL:

<http://yourhost/appname/Management.asmx>

Where *yourhost* and *appname* are the URL of the license server.

If you invoke the “Test” method, you should get an empty string back. If you see the string “error” it indicates that you do not have access to the licensing server. Add your computer’s IP address to the allowed list in the web.config file. If using role based security, be sure you use the IIS Management console to turn off anonymous authentication for the Management.asmx page.

If an error occurs, you can invoke the “Diagnostics” method to retrieve expanded trace diagnostics. In order to use this method, the *enablediagnostics* key under the *appSettings* section in your web.config file must be set to “true”. **Note that the web.config file now has**

diagnostics turned on by default. You need to disable this when done testing (look for the key <add key="enablediagnostics" value = "true" /> and set the value to "false").

Please include the text returned by the Diagnostics method with any support enquiries.

If you invoke the GetApplicationList method, you should get a list of applications on the licensing server (or none, if no applications are defined). This is an excellent method for verifying the internal operation and database access of the licensing server.

The License Manager and Development Components

The License Manager and Development Components are installed together. This is because developers need the License Manager to create the license files needed during the development process to run and test your licensed application. You can install the License Manager without the development sample code.

The installation sequence is quite simple. The only option that is important to consider is the default License Service URL which is set in the following dialog:

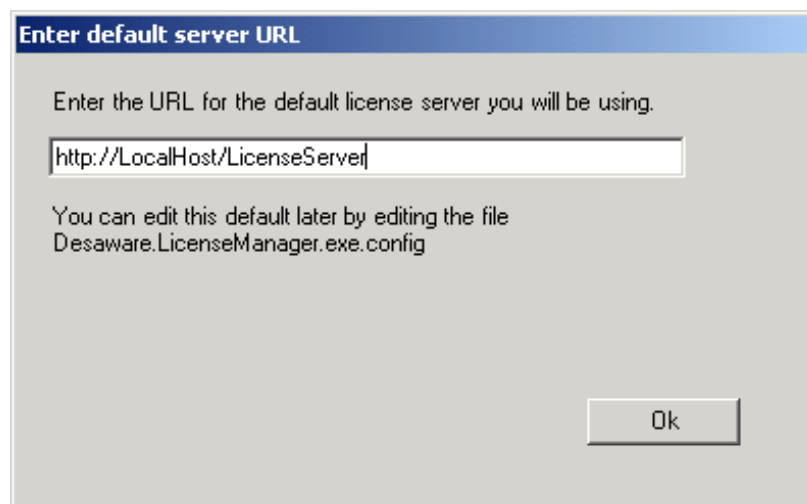


Figure 5
License Manager Installation Dialog Box

You can change the default server URL at any time by modifying the Desaware.LicenseManager.exe.config file as shown:

```
<appSettings>
<add key="Desaware.LicenseManager.Desaware.LicenseService.Management"
value="http://localhost/LicenseServer/management.asmx"/>
<add key="Desaware.LicenseManager20.Desaware.LicenseService.Management"
value="http://localhost/LicenseServer/management.asmx"/>
</appSettings>
```

If you specified a different port for the license server, use the standard URL port syntax. For example: to connect to a server on port 152, you would use the string:

```
http://LocalHost:152/LicenseServer
```

The License Manager

The License Manager application allows you to manage the licensing system. VB.NET source code for the License Manager is included (except for the DEMO edition). You may modify the License Manager for your internal use. It is also a good way to learn how to call the Licensing System's Web Service functions.

License Manager Security

The License Manager uses the Management.asmx entry point of the Licensing Server web service to perform most of its operations. Therefore, it is subject to whatever security constraints you placed on the Management.asmx entry point during installation or afterwards.

Knowing that one of the secrets of good security is to return as little information as possible when a security failure occurs, if the License Manager cannot connect to a service you will generally see very little. You will see no applications, and will not be able to create applications. There will be no additional error reports.

Important Security Note:

Communication between the License Manager and the Licensing Service is mostly unencrypted. This is because we assume that they are typically on a local Intranet. You should never use the License Manager to perform management tasks across the Internet unless through a secure VPN. If you wish to enable access through the Internet, we recommend you set the Management.asmx entry point to use SSL.

The License Manager will prompt for a user name and password if your service is secured using NTLM authentication. While the user name and password will be protected by the NTLM protocol, the actual data being sent to and from the service will be unencrypted unless you use SSL.

The Activator.asmx entry point, used by the client licensing component, does not require SSL because all significant data being passed to the service is encrypted.

Using the License Manager

The main License Manager form displays a list at the top of the form containing the Applications found for the currently connected licensing server. Under the Applications list on the left is a treeview control containing information related to the selected Application. Other controls will appear to the right of the treeview control depending on the information selected in the treeview control. You can select an Application by clicking under the Application Name heading in the Applications list. Most License Manager functions apply to the currently selected application. Also included in the main form are a number of menu commands to bring up the various forms that you will be using.

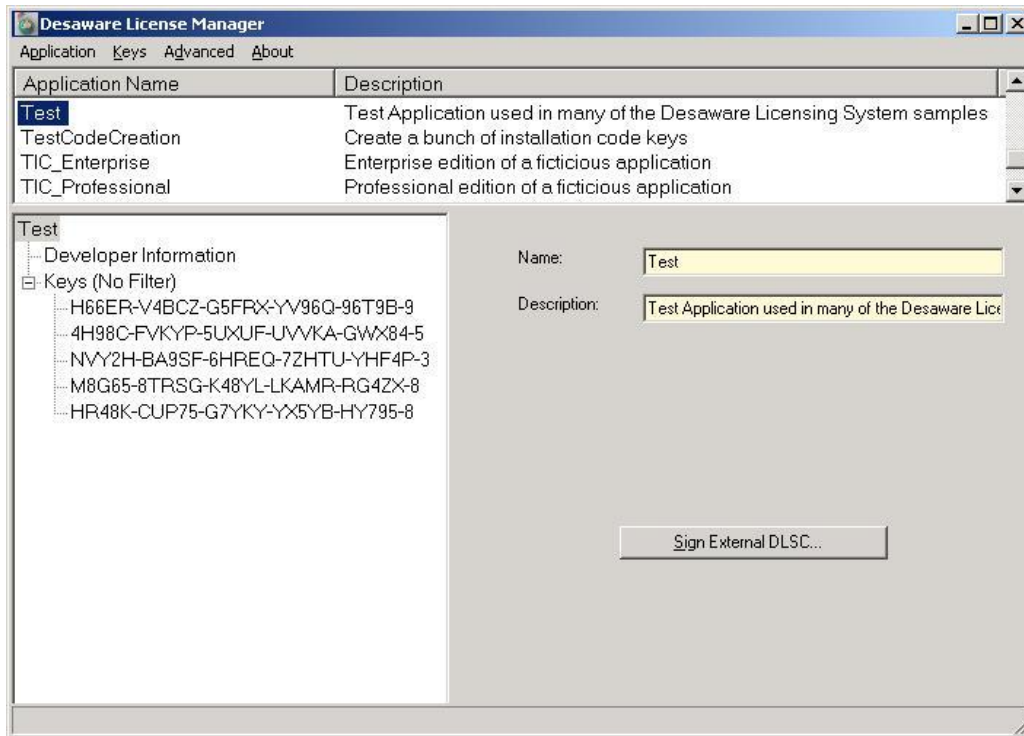


Figure 6
Main License Manager form

Menu – Advanced - Set Server Connection

The default licensing server is typically specified during installation of the License Manager. It is also specified in the file Desaware.LicenseManager.exe.config in the directory in which the License Manager was installed. Look for the appSettings section and set it as follows:

[VS .NET Framework version 2.0]

```
<appSettings>
<add key="Desaware.LicenseManager20.Desaware.LicenseService.Management"
value="http://localhost/LicenseServer/management.asmx"/>
</appSettings>
```

This dialog box allows you to select other license servers to manage. You must select a license server before proceeding.

The remaining Advanced menu entries will be described later.

Menu – Application - Create New Application

This dialog is used to create a new application. It is important that you understand the meaning of Application in the context of this product.

An Application defines all assemblies that are licensed together.

That means that different types of assemblies, different versions of one assembly, or assemblies across different products, can all be licensed with a single licensing certificate. For example: The MachineLicense component and the License Manager in this product are part of the same application. Minor revisions of these assemblies are still considered the same application.

As long as an assembly uses the same Application name, it will use the same DLSC licensing certificate, installation codes, etc.

The Single Application edition of the licensing system allows you to create only one application. However, you can upgrade the licensing system to a full version at a later time if you need to add additional applications.

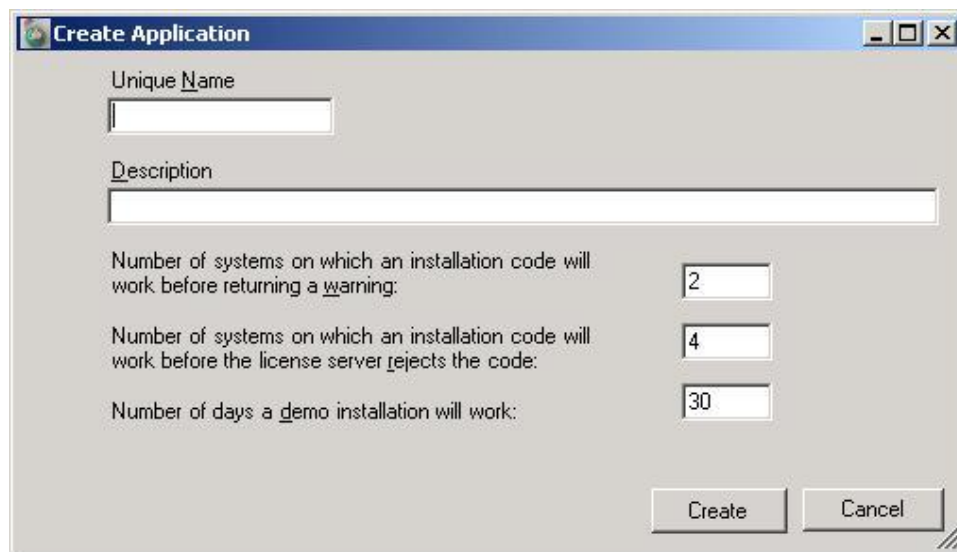


Figure 7
Creating an Application Dialog Box

Each application has a unique name. This name is limited to letters, numbers and the underscore character, and is limited to 50 characters, though you should keep it shorter. The Description is a description of the application. The Description can be up to 255 characters.

You can also specify the number of systems on which an Installation code will work before returning a warning to the client. The “Warning Count” is limited to a value from 1 to 100000.

Similarly, you specify the number of systems on which an Installation code will work before the license server rejects the code. The “Block Count” is limited to a value from 1 to 100000.

The Warning and Block status is returned to the client component during installation. Your software can deal with them as you choose.

Finally, you can specify the number of days a demo installation will work. If you do not wish to allow demo installations, set this value to zero. The “Demo Expiration” is limited to a value from 0 to 100000.

Note that just as you can license multiple assemblies as a single application, you can in fact use multiple applications in a single assembly. You might do this to license individual features of an application independently (refer to the ModuleLicensing sample project), or to license your application for a limited duration (refer to the “Subscription Application Note.pdf” file).

Developer Information

Select the Developer Information entry in the selected application treeview control to display information intended for use primarily by developers.

In order for clients to work with the licensing server, they need additional information beyond the application name. For example:

- The Application password (used if you allow installation without an Internet connection, or to do first pass verification if you do have an Internet connection).
- The demo expiration time for the application.
- The public key for the application.
- The URL of the licensing server.

Developers can use the *Copy password to clipboard* button to copy the Application password to the clipboard, or the *Copy Public Key to clipboard* button to copy the public key to the clipboard. However, the *Save info as ResX file* button makes this process even easier.

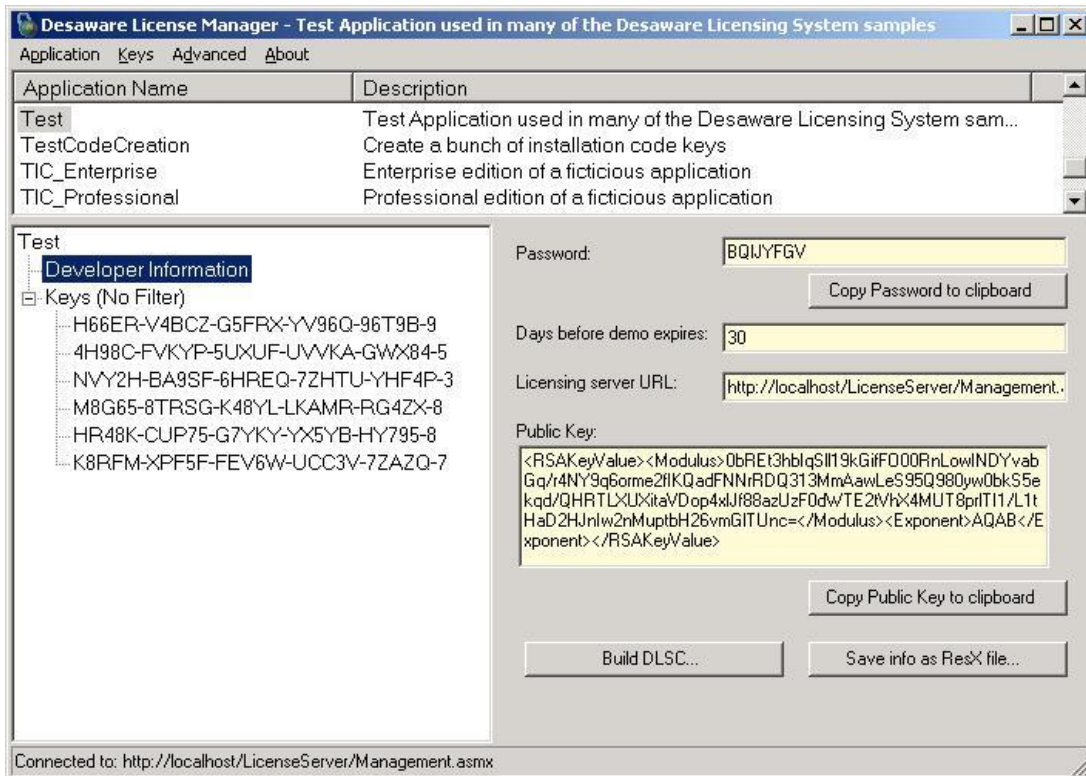


Figure 8
Displaying Developer Information

The *Save info as ResX file* button brings up a File Save dialog box that allows you to create an ResX resource file. Developers can add this file to a licensed application and use it to initialize the client licensing component with just one line of code. This is the preferred approach.

Developers also need a method to obtain a license certificate for use on their own system in order to develop and test with licensing before they've completed their installation program. The Build DLSC button creates a license certificate for the system on which the License Manager is running.

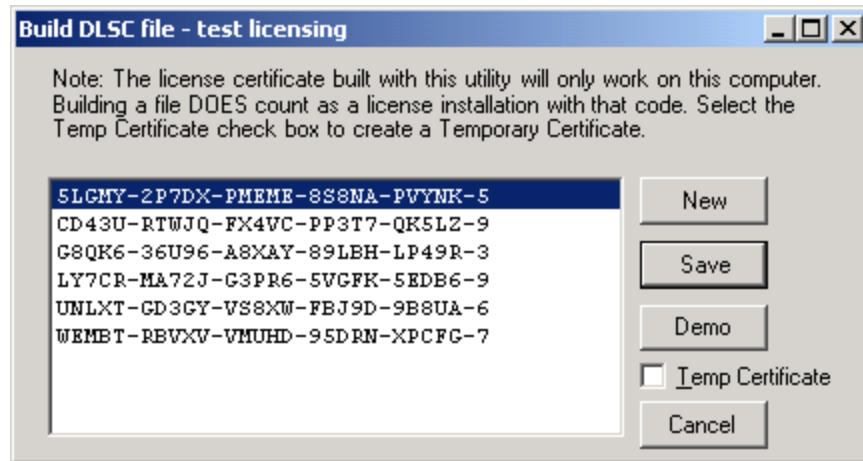


Figure 9
Build DLSC Option Dialog Box

The New command creates a new Installation Code and builds a DLSC file using that code.

The Load List command shows all existing installation codes for the application (it changes to the Save command after the codes are listed). Selecting the Save command builds a DLSC file using the selected code. This allows you to reuse existing codes and test reuse of codes on different machines.

The Demo command builds a Demo DLSC file (no installation code is used for demo installations).

The Temp Certificate checkbox builds a temporary license file (it tries to connect to an invalid server. Note that you will get an error message when you build a DLSC file with this option. This allows developers to test scenarios where temporary files are permitted.

Sign External DLSC

The *Sign External DLSC* button appears when the Application name is selected in the selected application treeview control (please refer to Figure 6). This option is designed for high security scenarios where an Internet connection is not available. In these cases you can configure your installation program to create a temporary certificate on the client's system, then have that certificate sent via email, or other secure channel to a local system. You can then use this option to digitally sign the certificate and then send it back to the client.

If the temporary certificate already contains an installation code, that code is used during the signing process. Any errors that occur are reported (for example, if the code or certificate is invalid).

If the temporary certificate does not contain an installation code, you will have the option of creating a new code, manually entering an installation code, or signing it without a code (in which case it will be a demo certificate).

The Single Application edition of the licensing system does not support this capability. However, you can upgrade the licensing system to a full version at a later time should you need to add this capability.

Menu - Keys - Create New Keys

This dialog box allows you to create one or more installation key codes. You can copy the new keys to the clipboard or save them to disk.

Note – The demo version of the Desaware Licensing System allows you to create up to 10 installation key codes per application. The Single Application edition allows you to create up to 1000 installation key codes.

Installation Keys

The installation Keys for the selected application are displayed in the treeview control. You can retrieve or update detailed installation information for a specific installation key code. Normally all key codes generated for a specific application are displayed as shown by the Keys (No Filter) entry. The License Manager provides several ways to filter key codes.

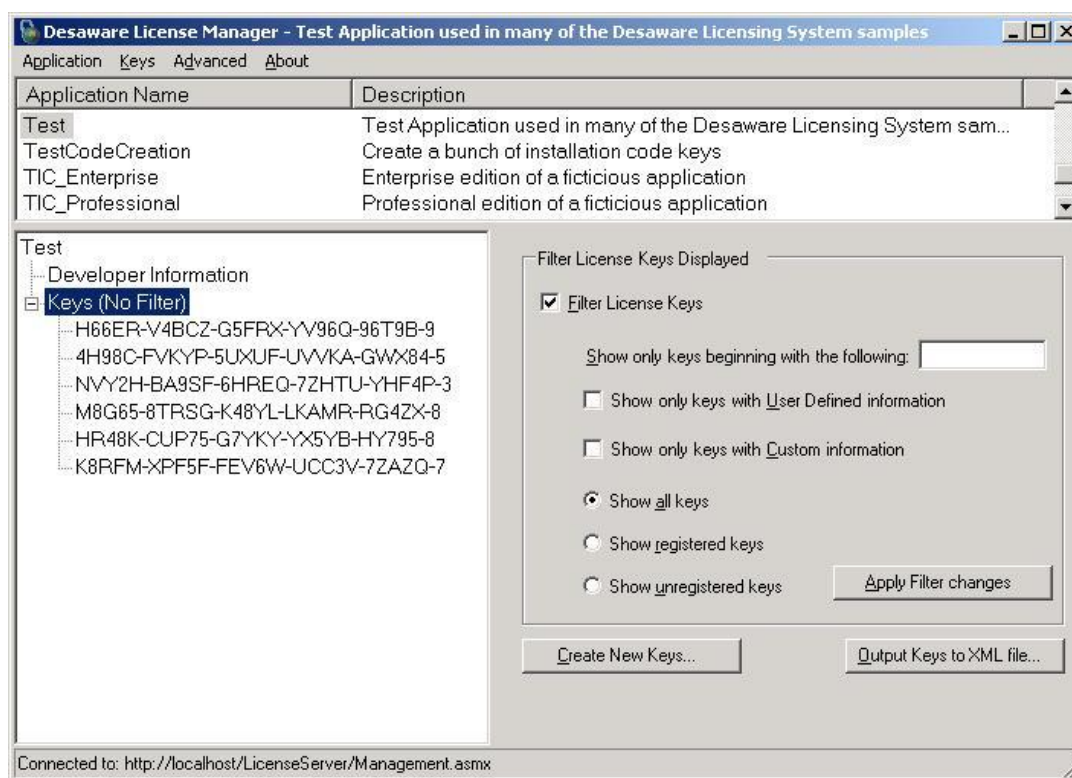


Figure 10
Installation Keys

The *Filter License Keys* check box enables/disables the rest of the filter controls. You can filter keys by entering their first characters or by displaying only keys containing specific information. You can specify a filter to display only keys starting with certain characters/digits by specifying the beginning of the key (up to 5 characters/digits) in the *Show only keys beginning with the following* text box. You can display only keys that contains User Defined information by selecting the *Show only keys with User Defined information* check box. You can display only

keys that contains Custom Data information by selecting the *Show only keys with Custom information* check box (this option is not available if the *Show unregistered keys* option button is selected since Custom information is only entered for registered keys). You can display the keys based on whether they have been registered (used) on your License Server or not by selecting one of the *Show keys* option buttons. After entering your filter criteria, click on the *Apply Filter changes* button to redisplay the installation keys based on your filters.

The *Create New Keys* button displays the *Create Keys* dialog allowing you to create new license keys for the selected application.

The *Output Keys to XML file* button will output detailed information for all of the keys displayed in the treeview control to an XML formatted text file. A sample output is displayed below:

```
<DesawareLicensingSystem>
<ServerURL>http://www.desaware.com/LicenseServer/management.asmx</ServerURL>
<ApplicationName>Test</ApplicationName>
<Keys>
  <HR48K-CUP75-G7YKY-YX5YB-HY795-8>
    <Count Warning="2" Block="4" />
    <UserDefined>Test changes</UserDefined>
    <a33eac1e-0ade-44ac-8dc0-5937a63435e4>
      <Date Installation="6/11/2003 12:00:00 AM" DemoExpiration="6/11/2003
12:00:00 AM" />
      <add key="name" value="Robert Harte">
        <add key="email" value="robertharte@somecompany.com">
          </a33eac1e-0ade-44ac-8dc0-5937a63435e4>
        <9eb5caa0-5ac9-4e67-a752-90cfb356d02c>
          <Date Installation="7/11/2003 12:00:00 AM" DemoExpiration="7/11/2003
12:00:00 AM" />
          <add key="name" value="William Pate" />
          </9eb5caa0-5ac9-4e67-a752-90cfb356d02c>
        </HR48K-CUP75-G7YKY-YX5YB-HY795-8>
      <K8RFM-XPF5F-FEV6W-UCC3V-7ZAZQ-7>
        <Count Warning="2" Block="4" />
      </K8RFM-XPF5F-FEV6W-UCC3V-7ZAZQ-7>
    </Keys>
  </DesawareLicensingSystem>
```

The *ServerURL* tag displays the License Server URL. The *ApplicationName* tag displays the selected application name. The *Keys* tag is the root node of a list of key nodes containing detailed information for each key. Each *key* node is identified by the actual license key code. Each *key* node always contains the *Warning* and *Block* count for that key. An optional *UserDefined* tag follows if that key contains any User Defined data. Another optional *UniqueInstallID* tag follows if that key was registered on the License Server. The tag key corresponds to the UniqueInstallID field in the License Server database. Within each UniqueInstallID node are keys indicating the InstallationDate, along with the DemoExpiration date, and any Custom Data information for that particular installation. Custom Data information are denoted by keys containing a “key-value” pair which corresponds to the field name and value of each custom data. Note that there could be multiple UniqueInstallID tags for each license key.

Menu – Keys - Find Key

This menu command displays the *Find Key* modeless dialog in which you can enter partial text of a license key code you wish to find within the keys treeview control. The key search is based on the currently displayed list of keys (filtered or not). The search is case insensitive, the '-' character may be used, and wildcards are not supported. The *Find* button performs the search from the beginning of the key treeview control, the *Find Next* button performs the search from the last matching key.

Individual Key information

Select an installation key code to display information for that particular key. You can update the *Attempts before warning given* and *Attempts before key is blocked* counts for the selected installation key code. For example: you sold a copy of your licensed product and the customer is so happy they wish to buy a site license for up to 100 developers. You can issue them 99 new installation key codes, or you can update the Warning and Block count for their existing code to 100 or higher, and allow them to use the same code on up to 100 machines.

There is a User defined information field of up to 255 characters associated with each key code that you can use for any purpose.

Click on the *Apply Changes* button to save the changes for the select key.

The *Show Detailed Info* button brings up a tree view for the selected installation key code showing each unique system on which the installation key code was used and the date on which it was installed (also the expiration of the demo if applicable). Custom data is also shown for the specified key.

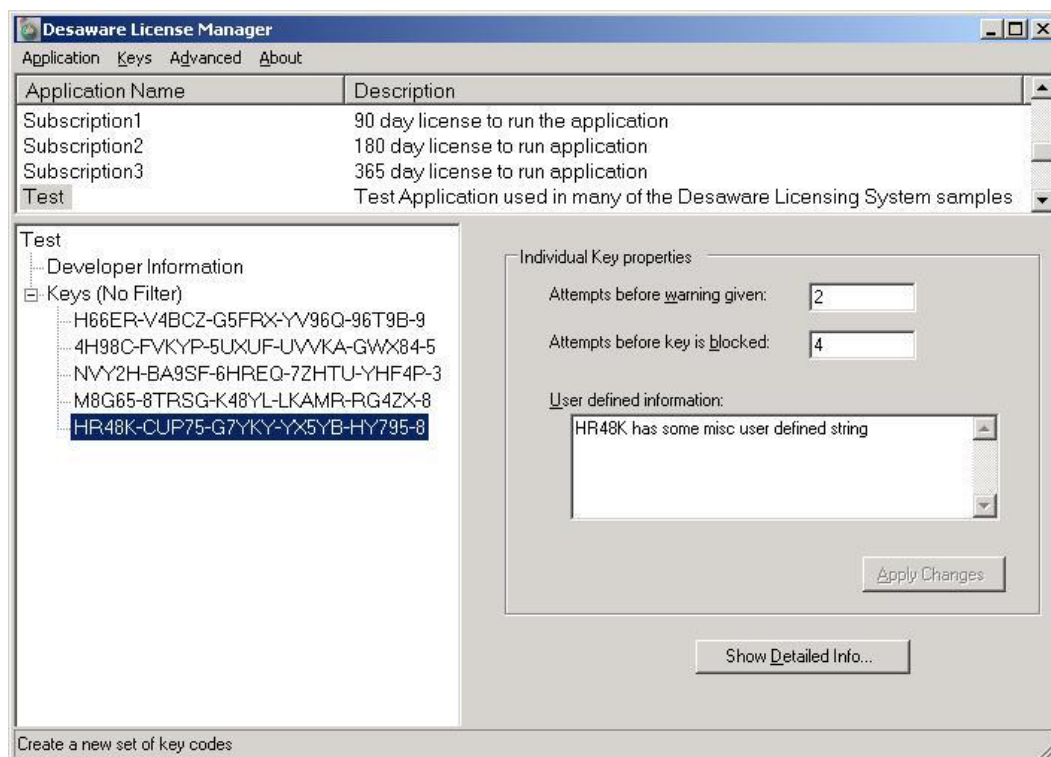


Figure 11 **Individual Key**

For each installation, you can view any custom data that was sent to the server during the installation. This might include registration information, for example.

Note, there is no facility in the license manager to modify custom information. The current version of the licensing server only provides a facility to capture this data from the client. You can, however, write your own utilities to work with the database to extract or modify this custom data.

Advanced License Manager Options (Hosted Installations)

This feature is not available in the DEMO edition of the Desaware Licensing System.

The following advanced options are generally used only for hosted installations. Normally, the installation, licensing and configuration of the server is handled by the installation program. However, if you are hosting your license server on a shared host (perhaps provided by an ISP or hosting service) it may not be possible to run the installation program. In these situations you must be sure that your hosting service allows you to configure the following:

1. Database support. This includes one of the following:
 - a. SQL server support, including the ability to assign database permissions to the account under which ASP.Net is running.
 - b. JET runtime support, including the ability to specify file and directory permissions to the account under which ASP.Net is running.
 - c. Other database support, in which you can grant the ASP.Net account sufficient permission to create tables and modify data in the required database.
2. Database creation. You must have the ability to define/create an empty database and specify a correct connection string for that database. Plus, you must be able to set the security for that database as specified previously.
3. Directory security. You must be able to set file/directory access security to allow the ASP.Net account to read or write files from directories you specify.
4. Code Access Security permissions. The Licensing Server requires full trust to run.

Refer to our online FAQ at <http://www.desaware.com/support/faq/licensing/index.aspx> for additional information on using the license server on shared hosts and to download the latest version of [the server configuration guide](#).

Security settings are both important and amongst the most challenging issues to manage. Your ASP.Net application runs under an account. That account can be specified in the system's machine.config file or in the Advanced configuration for the Application Pool hosting the application (Windows servers and IIS 7). By default the account is called ASPNET under Windows 2000, Network Service under XP and later. On server systems, you can specify the account used by the application pool in which the server is running. On Windows 7, the default setting is a special Application Pool Identity account which has the same name as the application pool. All references to "ASP.Net" account that follow refer to the account under which ASP.Net is running on your system.

To perform a hosted installation you must:

1. Install the license server on a local system in demo mode (in order to extract the necessary files, but not use the license key).
2. Create the appropriate virtual directory on the host system.
3. Copy the files to the host system. This includes the files in the root virtual directory and the bin directory.
4. If using Access, create a DB directory on the host system and set security for that directory to allow the ASP.Net account to read, write and modify files in that directory.

Create a blank Access database in that directory. We recommend using the IIS Management console to disable read access to this directory.

Be sure the database name and path matches that specified in the connection string.

5. If using SQL (or other database), create the necessary database and set security for the database to allow the ASP.Net account to create tables and perform all other access operations to that database.
6. Add the connection string and access type to the web.config file as described in the Appendix: "Configuring the Web Service web.config File". You must specify a valid database connection string, and the connection string must work under the account under which ASP.Net is running.
7. Temporarily set the security of the bin directory to allow the ASP.Net account to read and write files in the directory. (Note, if you are unable to do so, specify a directory that ASP.Net can write to in the appsettings section in your web.config file with the `alternateloadinstallpath` key: `<add key="alternateloadinstallpath" value="/writabledirectory" />`. The directory can be a physical directory on the system or relative to the web site root. The DLSC file will be placed in that directory and you should copy it to your bin directory after the hosted install is complete.
8. Use the License Manager Advanced— Set Server Connection menu command to select the correct server. You will get an error message "Server was unable to process request. --> The Desaware Licensing Service is not licensed for use on this server." - just ignore that error message since the server is not yet licensed.
9. Use the License Manager Advanced - Hosted Install menu command to enter your installation code for the server. This will license your hosted server.
10. Use the LicenseManager Advanced - Verify Hosted Database command to verify your connection to the database and initialize the database tables. Once this command succeeds, your license server is ready to use.
11. Reset the security of the bin directory to its original setting. If you used the `alternateloadinstallpath` key to place the DLSC file in a different directory, copy it to the bin directory.
12. If you are using role based security, use the IIS Management console to turn off basic authentication to the Management.asmx page.

Refer to the section “Testing and Debugging Server Installations” earlier in this document for assistance in resolving installation problems.

The Hosted Installation option is also used to upgrade the licensing system from the single application to full edition. Just enter your full version installation code to upgrade the selected server.

Advanced License Manager Options (Binding Code Generator)

This feature is not available in the DEMO edition of the Desaware Licensing System.

This feature is used to generate code that verifies your .NET assembly had not been modified. We highly recommend you read the accompanied StrongName.pdf document on using code binding to ensure that your .NET assemblies have not been modified.

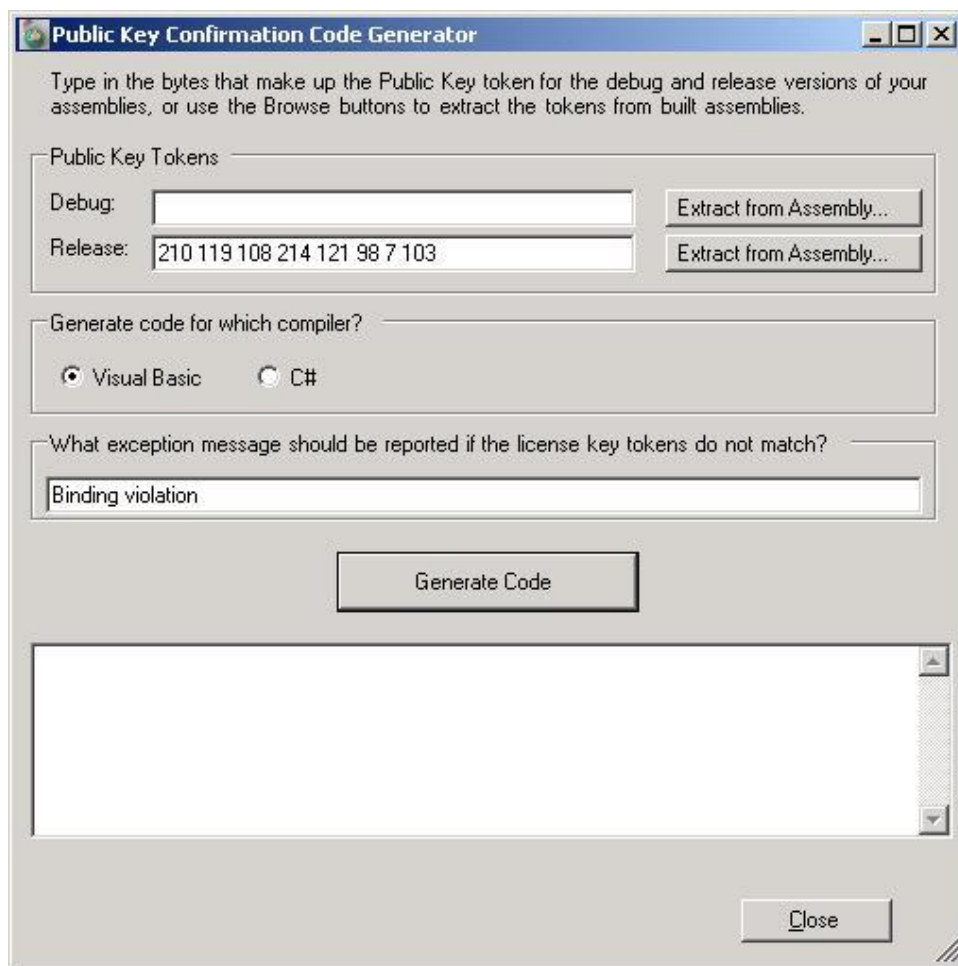
The image shows a Windows-style dialog box titled "Public Key Confirmation Code Generator". It contains several sections: 1. A title bar with standard window controls. 2. A text area with instructions: "Type in the bytes that make up the Public Key token for the debug and release versions of your assemblies, or use the Browse buttons to extract the tokens from built assemblies." 3. A section labeled "Public Key Tokens" with two text boxes: "Debug:" (empty) and "Release:" (containing "210 119 108 214 121 98 7 103"). Each text box has an "Extract from Assembly..." button to its right. 4. A section labeled "Generate code for which compiler?" with two radio buttons: "Visual Basic" (selected) and "C#". 5. A section labeled "What exception message should be reported if the license key tokens do not match?" with a text box containing "Binding violation". 6. A large "Generate Code" button. 7. A large empty text area at the bottom for output. 8. A "Close" button in the bottom right corner.

Figure 12
Binding Code Generator

Enter the public key tokens for your assembly in either of the Debug or Release text boxes. Separate each decimal value with a space. Or, you can select either of the Debug or Release *Extract from Assemblies* buttons to extract the public token key from a debug or release

compiled version of the assembly. Select the *Generate Code* button to generate the Visual Basic .NET or C# code for insertion into your application project.

Licensing Scenarios and Samples

The Desaware Licensing System supports a variety of licensing situations, but they will tend to fall into three different scenarios. We've included samples of how you might implement these three scenarios.

Important Notice! Warning! Danger!

If you implement a scenario that requires activation for your software to install, and at some future date your licensing server becomes unavailable, that software will become uninstallable. It will not be possible to enable features for which you choose to require activation.

This system uses 128 bit encryption, and there are no backdoors or secret codes built in that will allow you to bypass this licensing scheme. Neither do we know of any flaws at this time that might allow you to do so.

We ourselves have no tools or technology that would allow recovery or regeneration of the private key necessary to re-enable licensing of software if the original license server database is lost.

We encourage you to choose the “Friendly” scenario if you wish you allow your software to be installable/runable without server activation (even though it does not provide the same level of security).

On Strong Names

Important Notice!

Licensed assemblies should always be strong-named. If you do not use a strong name, it will be possible for people to bypass your licensing code by modifying your assembly or using other methods. Strong naming an assembly both protects your code from modification, and cryptographically binds it to the licensing system. We highly recommend that you refer to the separate “StrongName.pdf” article (*included in the full edition of the Desaware Licensing System*) written by Dan Appleman for more details on protecting your assemblies.

You should also use obfuscation (*the Desaware QND Obfuscator is included in the full edition of the Desaware Licensing System*) to help protect your assembly from other forms of attack including modification of internal variables via reflection, or complete disassembly/reassembly of your code.

Preparing to Use the Samples

Before you can try any of the sample programs, you must do the following:

1. Install the Licensing Server on the system of your choice as described in the directions earlier. Be sure to allow access to the Management.asmx page from the development machine on which you are testing the software.
2. Install the developer/management software. The server and development/management software can be installed on the same system.
3. Using the License Manager, create an application. We will use an application named “Test” for several of the samples.
4. Using the Application-Developer Info dialog box, use the Save ResX command to create the Test.resx file for your server.
5. Copy the new Test.resx file to replace the test.resx file included with the sample programs. **If you use the test.resx file we provide, you will (correctly) see licensing failures, because they contain information for our own test server!**
6. Use the Installation Code-Create Codes command to create several installation codes to use.
7. Copy your Desaware.DLS10Client.dll file to the executable directory for the sample. This is required to debug the sample.

The sample code shown here assumes that the module has an Imports (VB) or using (C#) reference to the Desaware.MachineLicense namespace.

High Security Scenario

This scenario represents the highest possible level of security. It has the following characteristics:

- End to end cryptographically secure.
- Requires an Internet connection and activation.
- Precise limits on the number of systems on which an installation code can be used.
- If your licensing server is down or unavailable, the software cannot be installed (note, you can list multiple license servers in the software you distribute).

This scenario is called “*helping our customers stay honest*” by some, “*we don’t trust our customers one bit*” by others. We don’t use it ourselves, but making sure there was a high security scenario was a top design priority. Why? Because it is infinitely better to start with tight security and selectively relax it, than to start with relaxed security and try to make it tighter.

Implementing the High Security Scenario

One of the curious contradictions about security systems is this: higher security tends to come from simplicity, not complexity. As you reduce options and features, you have fewer variations to deal with, and fewer potential attack points.

Thus it turns out that the high security scenario is the simplest to implement. How simple? You'll be amazed.

Creating a Licensed Application

1. Create a new assembly to be licensed.
2. Reference the Desaware.MachineLicense.dll component in your project.
3. Add a ClientLicense component to the project. This can be done by dragging it from the toolbar onto a form or component container, or using code (as shown shortly).
You can add the ClientLicense component to your toolbox by right clicking on the toolbox and selecting the Customize Toolbox menu item. Then select the .NET Framework Components tab and check the ClientLicense checkbox.
If the ClientLicense checkbox does not appear, click on the Browse button and select the Desaware.MachineLicense.dll file to add the ClientLicense component to the list of .NET components.
4. As noted at the beginning of this section, you should have used the License Manager application to create the Resx resource file (test.resx). Add this file to the project.
5. Set the ResourceName property of the ClientLicense component to the name of the resource file (without the .Resx extension).

In code, you can accomplish the same effect as follows:

[VB]

```
Dim ClientLicense1 As New Desaware.MachineLicense.ClientLicense("Test")
```

[C#]

```
internal Desaware.MachineLicense.ClientLicense ClientLicense1;  
ClientLicense1 = new Desaware.MachineLicense.ClientLicense("Test");
```

Verifying if an Assembly is Licensed

The HighSecurity sample application performs a license validation during the form load event. This demonstrates the simplest type of verification:

[VB]

```
Private Sub Form1_Load(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles MyBase.Load  
    ClientLicense1.VerifyLicense(True)  
    If ClientLicense1.Licensed Then  
        lblLicensed.Text = "Application is licensed"  
    Else  
        If ClientLicense1.DemoVersion Then  
            lblLicensed.Text = "Application is a demo version"  
        Else  
            lblLicensed.Text = "Application is not licensed"  
        End If  
    End If  
End Sub
```

[C#]

```
private void Form1_Load(object sender, System.EventArgs e)
{
    ClientLicense1.VerifyLicense(true);
    if (ClientLicense1.Licensed)
    {
        lblLicensed.Text = "Application is licensed";
    }
    else
    {
        if (ClientLicense1.DemoVersion)
        {
            lblLicensed.Text = "Application is a demo version";
        }
        else
        {
            lblLicensed.Text = "Application is not licensed";
        }
    }
}
```

That's it. One line to create and initialize the component. One line to do the verification.

Two properties, `Licensed` and `DemoVersion`, to determine if the license is valid.

There are some additional properties, as you'll see later, that allow you to provide additional feedback as to why the license failed, but for most situations, this is all the code you'll need for the High Security scenario.

Licensing the Assembly

The first time you run the sample application it will, of course, display that it is unlicensed. That's because you never created a license file for the application.

There are two types of licenses, a demo license and a full license – the difference between them being that a demo license expires after a set time, and does not have an installation code.

A license can be created by a separate program (i.e., your installer program), or by your own program (in other words, you can allow your demo version to be converted into a full version by allowing the user to activate the license from within the program). That's how the HighSecurity example does it.

Drop a `CodeEntryControl` on the form (for details on how to do this, refer to the earlier **Creating a Licensed Application** section and follow similar steps as adding the `ClientLicense` control). This is a control designed to make it easy to enter a control code.

In the next example, you'll learn how you can create the `CodeEntryControl` to provide first pass verification of an installation code and let the user know when a valid installation code has been entered, however in this example we'll keep things simple.

If an installation code has been entered (valid or not), clicking the Install button will cause a license to be created if possible. Otherwise clicking the install button will cause a demo license to be created.

[VB]

```
Private Sub cmdInstall_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles cmdInstall.Click
    Dim results As InstallErrorResults
    If CodeEntryControl1.InstallCode = "" Then
        results = ClientLicense1.InstallDemo( _
            InstallationModes.SyncActivationRequired)
    Else
        results = ClientLicense1.InstallLicense( _
            CodeEntryControl1.InstallCode, _
            InstallationModes.SyncActivationRequired)
    End If
    If results > InstallErrorResults.FatalError Then
        MsgBox(ClientLicense1.GetResultDescription(results))
    End If
End Sub
```

[C#]

```
private void cmdInstall_Click(object sender, System.EventArgs e)
{
    InstallErrorResults results;
    if (CodeEntryControl1.InstallCode == "")
    {
        results = ClientLicense1.InstallDemo(Desaware.MachineLicense.
            InstallationModes.SyncActivationRequired);
    }
    else
    {
        results = ClientLicense1.InstallLicense(CodeEntryControl1.
            InstallCode, Desaware.MachineLicense.InstallationModes.
            SyncActivationRequired);
    }
    if (results > InstallErrorResults.FatalError)
    {
        System.Windows.Forms.MessageBox.Show
            (ClientLicense1.GetResultDescription(results));
    }
}
```

Try clicking on the Install button without entering a code. If you have things set up correctly, the next time you run the program, it should report that it is validated as a demo version of the program.

Now try entering a random installation code and clicking the install button. You'll get an Invalid installation code error.

Now enter a valid installation code (obtained from the License Manager application) and click on the Install button.

Next time you run the program you will see that it is fully licensed.

The ClientLicense InstallDemo and InstallLicense methods return specific information on the installation process. Numbers lower than FatalError (99) represent success, and offer additional information that you may wish to use in customizing your user interface. For example: If an installation was allowed, but exceeds the “Warning” number for the application, you may wish to let the user know that there is a potential license violation, because the installation code has been used on multiple systems.

If the user has just installed a demo license on a system that the server detects as already having a full license, the demo license is installed, but you may wish to notify the user of the fact that they have already used an installation code on that system and may wish to reuse it to reenable the full license. There are additional results that do not apply to the high security scenario.

That’s it. You can add additional user feedback, and of course have to decide exactly what to do in your assembly if it is a demo version or unlicensed. And you’ll shortly see how to improve the use of the CodeEntryControl to provide quick feedback on installation code errors. But the licensing itself is just that easy.

If you wish to repeat the experiment, just delete the license file (in this case Test.dlsc) that is in the assembly’s directory.

Variations

The majority of the variations of this scenario relate to the user interface and deciding how the application should work in unlicensed or demo mode. Other options to consider include:

- Sending additional information (such as registration information) to the server during activation.
- Modifying the System Matching algorithm or adding additional system identifiers (as described later).
- Requiring an internet connection in order to run in demo mode. Setting the ClientLicense’s DemoRequireInternet property will attempt to retrieve the current date from a number of internet time servers when the demo license verification is performed. The date retrieved will be used (rather than using the system date) to determine whether the demo has expired.

Potential Vulnerabilities

- You can get licensed software to run on more than one system by matching one of the system identifiers so that the licensing server thinks they are the same system. You can add your own custom matching algorithm to tighten security even further, for example: by locking the software to a specific network card or CPU serial number.
- Refer to the section “Code Reuse Blocking” for unusual cases that can lead to extra authorizations of an installation code.

- If demos are permitted to run without an Internet connection, the user can set back their system date to extend demo periods.

More “Friendly” Scenarios

This scenario is forgiving of cases where an Internet connection or the licensing server is unavailable. It has the following characteristics:

- Cryptographically secure only if an Internet connection exists.
- Will attempt a deferred connection to the licensing server once a connection exists. You decide what to do in case the license is invalid.
- Good security even if an Internet connection does not exist.

This scenario is for those who do trust their customers to be honest. It provides outstanding license control as long as the customer does not intentionally work to subvert the licensing.

Secure licensing requires connection to a server. Any licensing system that does not include connection to a server is based on “secrets” – hidden files, encryption, registry entries, etc. – all of which can be bypassed or defeated with enough effort.

One of the primary design goals for the Desaware Licensing System was to allow licensing without an Internet connection. The approach we took is as follows:

- If an Internet connection exists, licensing proceeds exactly as described in the High Security scenario.
- If an Internet connection does not exist, we use hidden information (the application password) to verify an installation code and create a temporary certificate.
- During verification, if an Internet connection exists, the licensing component will send the temporary certificate to the server for signing. The results of that attempt are made available to your application to handle as you wish.

Implementing the Friendly Security Scenario

Because the Friendly Security scenario involves additional flexibility, it is somewhat more complex and offers more variations.

Creating a Licensed Application

1. Create a new assembly to be licensed.
2. Reference the Desaware.MachineLicense.dll component in your project.
3. Add a ClientLicense component to the project. This can be done by dragging it from the toolbar onto a form or component container, or using code (as shown shortly).

You can add the ClientLicense component to your toolbox by right clicking on the toolbox and selecting the Customize Toolbox menu item. Select the .NET Framework Components tab and check the ClientLicense checkbox.

If the ClientLicense checkbox does not appear, click on the Browse button and select the

Desaware.MachineLicense.dll file to add the ClientLicense component to the list of .NET components.

4. As noted at the beginning of this section, you should have used the License Manager application to create the Resx resource file (test.resx). Add this file to the project.
5. Set the ResourceName property of the ClientLicense component to the name of the resource file (without the .Resx extension).

In code, you can accomplish the same effect as follows:

[VB]

```
Dim WithEvents ClientLicense1 As New _  
Desaware.MachineLicense.ClientLicense("Test")
```

[C#]

```
internal Desaware.MachineLicense.ClientLicense ClientLicense1;  
ClientLicense1 = new Desaware.MachineLicense.ClientLicense("Test");
```

Verifying if an Assembly is Licensed

The FriendlySecurity sample application performs a license validation during the form load event. This demonstrates a slightly more complex form of verification:

[VB]

```
Dim vs As ValidationStatus  
CodeEntryControl1.License = ClientLicense1  
vs = ClientLicense1.VerifyLicense(True)  
If ClientLicense1.Licensed Then  
    lblLicensed.Text = "Application is licensed"  
Else  
    If ClientLicense1.DemoVersion Then  
        lblLicensed.Text = "Application is a demo version, " & _  
            & ClientLicense1.DemoDaysRemaining.ToString() & _  
            " Days remaining"  
    Else  
        lblLicensed.Text = "Application is not licensed"  
    End If  
End If  
If ClientLicense1.Licensed OrElse ClientLicense1.DemoVersion Then  
    If ClientLicense1.DeferredLicense Then  
        lblTemp.Text = "License is based on a temp certificate"  
    Else  
        lblTemp.Text = "License is based on a signed certificate"  
    End If  
End If
```

[C#]

```
ValidationStatus vs;  
  
CodeEntryControl1.License = ClientLicense1;
```

```

vs = ClientLicense1.VerifyLicense(true);

if (ClientLicense1.Licensed)
{
    lblLicensed.Text = "Application is licensed";
}
else
{
    if (ClientLicense1.DemoVersion)
    {
        lblLicensed.Text = "Application is a demo version, " +
            ClientLicense1.DemoDaysRemaining.ToString() +
            " Days remaining";
    }
    else
    {
        lblLicensed.Text = "Application is not licensed";
    }
}
if ((ClientLicense1.Licensed) || (ClientLicense1.DemoVersion))
{
    if (ClientLicense1.DeferredLicense)
    {
        lblTemp.Text = "License is based on a temp certificate";
    }
    else
    {
        lblTemp.Text = "License is based on a signed certificate";
    }
}

```

As you can see, there is one additional property to consider in this case. The `DeferredLicense` property indicates that the licensing results are based on a temporary (unsigned) license. It is up to you to decide whether to change the behavior of your component based on whether it is using a temporary or signed license.

Note, this example also illustrates how you can determine the number of days left in a demo installation.

Using the CodeEntryControl Control

Please be sure you read the “Licensing the Assembly” section for the High Security scenario before continuing.

This scenario demonstrates a more advanced use of the `CodeEntryControl` control. Instead of a single command button, there are two, one for a regular install, the other for a demo install. The regular install command button is disabled by default.

In the `Form_Load` event, the `License` property for the `CodeEntryControl` control is set to the `ClientLicense` in use using the following code:

```
CodeEntryControl1.License = ClientLicense1
```

This allows the `CodeEntryControl` to do first pass verification of the installation code. The `LicenseCodeEntered` event is raised whenever the status of the code in the control changes.

[VB]

```
Private Sub CodeEntryControl1_LicenseCodeEntered(ByVal Sender As _  
Object, ByVal e As Desaware.MachineLicense.CodeEntryEventArgs) _  
Handles CodeEntryControl1.LicenseCodeEntered  
    cmdInstall.Enabled = e.IsValid  
End Sub
```

[C#]

```
private void CodeEntryControl1_LicenseCodeEntered(object Sender,  
    Desaware.MachineLicense.CodeEntryEventArgs e)  
{  
    cmdInstall.Enabled = e.IsValid;  
}
```

The regular install command button will only be enabled when there is a valid code in the control.

If you do not set the License property of the CodeEntryControl, the IsValid property of the CodeEntryEventArgs will be True any time there are 26 characters in the control – no verification will take place. The CodeEntryEventArgs object includes a ValidationChecked property to let you know whether validation took place.

Note: The CodeEntryControl.InstallCode property will be an empty string if 26 characters are not present (no validation) or if the code is not valid (with validation).

Licensing the Assembly

The licensing code for this scenario is virtually identical to the High Security scenario. The main difference is the use of the SyncAllowDeferred argument instead of the SyncActivationRequired argument in the InstallLicense and InstallDemo methods. This instructs the licensing component to install a temporary certificate if a licensing server cannot be reached.

This example also has separate methods for the full and demo install, but that is a variation that can be applied to any licensing scenario.

[VB]

```
Private Sub cmdInstall_Click(ByVal sender As System.Object, _  
ByVal e As System.EventArgs) Handles cmdInstall.Click  
    Dim results As InstallErrorResults  
    results = ClientLicense1.InstallLicense( _  
        CodeEntryControl1.InstallCode, _  
        InstallationModes.SyncAllowDeferred)  
    If (Not ClientLicense1.DeferredLicense) AndAlso results >  
InstallErrorResults.FatalError Then  
        MsgBox(ClientLicense1.GetResultDescription(results))  
    End If  
End Sub  
  
Private Sub cmdDemo_Click(ByVal sender As System.Object, _
```

```

ByVal e As System.EventArgs) Handles cmdDemo.Click
    Dim results As InstallErrorResults
    results = _
        ClientLicense1.InstallDemo(InstallationModes.SyncAllowDeferred)
    If (Not ClientLicense1.DeferredLicense) AndAlso results >
        InstallErrorResults.FatalError Then
        MsgBox(ClientLicense1.GetResultDescription(results))
    End If
End Sub

```

[C#]

```

private void cmdInstall_Click(object sender, System.EventArgs e)
{
    InstallErrorResults results;

    results = ClientLicense1.InstallLicense(
        CodeEntryControll1.InstallCode, Desaware.MachineLicense.
        InstallationModes.SyncAllowDeferred);
    if ((!ClientLicense1.DeferredLicense) &&
        (results > InstallErrorResults.FatalError))
    {
        System.Windows.Forms.MessageBox.Show(
            ClientLicense1.GetResultDescription(results));
    }
}

private void cmdDemo_Click(object sender, System.EventArgs e)
{
    InstallErrorResults results;

    results = ClientLicense1.InstallDemo(Desaware.MachineLicense.
        InstallationModes.SyncAllowDeferred);
    if ((!ClientLicense1.DeferredLicense) &&
        (results > InstallErrorResults.FatalError))
    {
        System.Windows.Forms.MessageBox.Show(
            ClientLicense1.GetResultDescription(results));
    }
}

```

If the `DeferredLicense` property is `True`, the currently installed license is a temporary license – meaning an error occurred (such as the server couldn't be reached) that did not indicate the license was invalid. In other words, if the result from the server is that the installation code is invalid or blocked, no temporary certificate would be installed, so the `DeferredLicense` property would be `false`.

You can also use the `AsyncAllowDeferred` option for installation. This option immediately installs a temporary license, then proceeds to attempt to obtain a signed license in a background thread.

Handling Deferred Licensing Results

When you use the `SyncAllowDeferred` option and the server cannot be reached, the licensing component will attempt to obtain a signed license each time you call the `VerifyLicense` method.

The VerifyLicense method takes a single parameter “DeferredCheckInBackGround”, which causes the license check to happen asynchronously – this is the preferred approach.

The AsyncAllowDeferred installation option also causes the server connection to occur in the background.

Regardless of how the background installation is triggered, the response is the same. The ClientLicense component will raise a DeferredInstallComplete event when the attempt to install is complete. The ClientLicenseEventArgs event parameter contains the value InstallErrorResults that indicates the result of the installation. This value can also be obtained from the ClientLicense InstallError property.

[VB]

```
Private Delegate Sub UserWarningDelegate(ByVal warning As String)

Private Sub UserWarning(ByVal warning As String)
    MsgBox("An error occurred during installation: " & warning)
End Sub

Private Sub ClientLicense1_DeferredInstallComplete(ByVal Sender _
As Object, ByVal e As Desaware.MachineLicense.ClientLicenseEventArgs) _
Handles ClientLicense1.DeferredInstallComplete
    If e.InstallResults > InstallErrorResults.FatalError Then
        Dim userdelegate As UserWarningDelegate = _
            AddressOf Me.UserWarning
        Me.Invoke(userdelegate, New Object() _
            {ClientLicense1.GetResultDescription(e.InstallResults)})
    End If
End Sub
```

[C#]

```
private delegate void UserWarningDelegate(string warning);

private void UserWarning(string warning)
{
    System.Windows.Forms.MessageBox.Show ("An error occurred
        during installation: " + warning);
}

private void ClientLicense1_DeferredInstallComplete(object Sender,
Desaware.MachineLicense.ClientLicenseEventArgs e)
{
    if (e.InstallResults > InstallErrorResults.FatalError)
    {
        UserWarningDelegate userdelegate = new
            UserWarningDelegate(this.UserWarning);
        this.Invoke(userdelegate, new object[]
            {ClientLicense1.GetResultDescription(e.InstallResults)});
    }
}
```

In this example, when the asynchronous operation returns a fatal error, it brings up a message box to the user. Note that the event occurs on a background thread, thus communication through the main form must be through the `Invoke` method.

This brings you to the most important question you need to answer. What do you do if a temporary license was installed, and the background installation indicates that a license violation has occurred?

By default, if a fatal error occurs during a background operation, the licensing component does the following:

- If a temporary certificate had just been installed (as occurs when doing an install), the certificate will be deleted if (and only if) the server responds with a specific license violation, such as an invalid installation code, blocked installation code, or expired demo. It will not delete the temporary certificate if the server is unreachable, does not support that application, or has some other error.
- If the background operation was triggered by a `VerifyLicense` command, the license component will NOT delete an existing temporary certificate, even if a fatal error occurs. This follows the general principle of not disabling a working application.

You can delete the license at any time. The license can be found by querying the `ClientLicense` component's `LicenseFilePath` property.

Variations

As before, most of the variations on this scenario relate to the user interface and deciding how the application should work in unlicensed or demo mode. Other options to consider include:

- What to do if during a verification you discover that the temporary license is valid, but the server reports a license violation?
- Sending additional information (such as registration information) to the server during activation.
- Retrieving server data (described later) from the license file.
- Modifying the System Matching algorithm or adding additional system identifiers (described later).

The Desaware Variation

Our own licensing approach is based on the “Friendly” scenario, with the following additions:

- The only time we disable working code is when a demo version expires.
- We allow one or two “extra” uses of an installation code to add tolerance for cases where the software is uninstalled, or accidentally installed on the wrong machine.
- We do not disable a working temporary license.

Potential Vulnerabilities:

- You can get licensed software to run on more than one system by matching one of the system identifiers so that the licensing server thinks they are the same system. You can add your own custom matching algorithm to tighten security further, for example: by locking the software to a specific network card or CPU serial number.
- Because licensing is permitted without an Internet connection, you can install the software while the computer is unplugged from the Internet and it will run. License verification will proceed after an Internet connection is available. You have great flexibility in deciding what to do when running under a temporary license, and what to do if such license turns out to be invalid - thus you can control the level of security.
- If demos are permitted to run without an Internet connection, the user can set back their system date to extend demo periods.
- Refer to the section “Code Reuse Blocking” for unusual cases that can lead to extra authorizations of an installation code.

The InstallerLicenseTool Example

The InstallerLicenseTool example demonstrates a slightly more sophisticated example of using the licensing component. It demonstrates:

- Capturing registration data to the server.
- Use of the licensing component as part of a Windows Installer custom tool.

You may recognize the sample. It’s the same code we use during installation of the Desaware Licensing System.

Licensing Components

The .NET framework includes a set of classes that implement licensing for components. The first thing you need to know about this framework is that you don’t need to use it.

If you have a component, such as a business object or web service or class library – where there is no distinction between design time and runtime - simply use the licensing component as shown earlier, and throw an exception (or do whatever you feel is appropriate) if your component is unlicensed.

The .NET Framework licensing classes implement a fairly traditional form of component licensing – where the host notifies the component if it is in design mode or run mode. While in design mode, the design environment retrieves a licensing string which it passes to the component for verification at runtime.

While it can be extended in various ways, the base scenario is to make sure anyone using a component in their projects is properly licensed, and to allow the component to run embedded within that project without requiring the user of that component to be licensed. In other words, the developer of the component buys the license, but people running applications built with that component do not. The default license provider (LicFileProvider) that comes with .NET looks for a “license file” (aka text file) in the directory of the component, and uses the contents of that file as a license key.

The Desaware Licensing System includes a LicenseProvider that allows you to adopt this approach, with considerably more security.

The Components sample directory includes two projects: LicensedControl and UsesLicensedControl that demonstrate use of the DlsLicenseProvider class.

To do so, you do the following:

1. Add the LicenseProvider attribute to your component and implement the IDlsLicensedComponent interface as shown here:

[VB]

```
<System.ComponentModel.LicenseProvider(GetType(_  
Desaware.MachineLicense.DlsProvider))> Public Class LicensedControl1  
    Inherits System.Windows.Forms.UserControl  
    Implements IDlsLicensedComponent
```

[C#]

```
[System.ComponentModel.LicenseProvider(typeof(Desaware.MachineLicense.  
DlsProvider))]public class LicenseControl1 :  
    System.Windows.Forms.UserControl, IDlsLicensedComponent
```

Create an instance of the ClientLicense component. Don't just drop it on the control's form. Why? Because licensing of components is done before the controls on the component are instantiated, so if you just drop it on the form it won't actually exist during license verification.

Be sure you added the resource file Test.resx to the component being licensed!

[VB]

```
Private ClientLicense1 As New ClientLicense("Test")
```

[C#]

```
private ClientLicense ClientLicense1 = new ClientLicense("Test");
```

Implement the IDlsLicensedComponent interface. The LicenseProvider reflects its request for license keys back to the component. This allows you to make your own logic choices in terms of when a component should or should not be licensed.

[VB]

```
Public Function GetLicenseKey(ByVal usage As _  
System.ComponentModel.LicenseUsageMode) As String _  
Implements IDlsLicensedComponent.GetLicenseKey  
    VerifyLicense()  
    Return ClientLicense1.RuntimeKey  
End Function  
  
Public Function VerifyLicenseKey(ByVal RuntimeKey As String) _
```

```

As Boolean Implements IDlsLicensedComponent.VerifyLicenseKey
    Dim result As Boolean
    result = ClientLicense1.VerifyRuntimeKey(RuntimeKey)
    Return result
End Function

```

```

Private Sub VerifyLicense()
    Static VerificationTested As Boolean
    If Not VerificationTested Then
        ClientLicense1.VerifyLicense(True)
        VerificationTested = True
    End If
End Sub

```

[C#]

```

public string GetLicenseKey(System.ComponentModel.
    LicenseUsageMode usage)
{
    VerifyLicense();
    return ClientLicense1.RuntimeKey;
}

public bool VerifyLicenseKey(string RuntimeKey)
{
    bool result;
    result = ClientLicense1.VerifyRuntimeKey(RuntimeKey);
    return result;
}

private void VerifyLicense()
{
    if (!VerificationTested)
    {
        ClientLicense1.VerifyLicense(true);
        VerificationTested = true;
    }
}

```

The GetLicenseKey function should, by default, return the value of the ClientLicense1 RuntimeKey property. This can be verified in the VerifyLicenseKey function using the VerifyRuntimeKey method.

The following code simply allows the control to display its current licensing status.

[VB]

```

Private Sub LicensedControl1_Load(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
    VerifyLicense()
    lblLicensed.Location = New Point(0, 0)
    lblLicensed.Size = New Size(Width, Height)

```

```

If ClientLicense1.Licensed Then
    lblLicensed.Text = "Full license"
Else
    If ClientLicense1.DemoVersion Then
        lblLicensed.Text = "Demo license"
    Else
        lblLicensed.Text = "No license"
    End If
End If
If ClientLicense1.DeferredLicense Then lblLicensed.Text _
= lblLicensed.Text & " - temp"
End Sub

```

[C#]

```

private void LicenseControll_Load(object sender, System.EventArgs e)
{
    VerifyLicense();
    lblLicensed.Location = new Point(0, 0);
    lblLicensed.Size = new Size(Width, Height);

    if (ClientLicense1.Licensed)
        lblLicensed.Text = "Full license";
    else
    {
        if (ClientLicense1.DemoVersion)
            lblLicensed.Text = "Demo license";
        else
            lblLicensed.Text = "No license";
    }
    if (ClientLicense1.DeferredLicense)
        lblLicensed.Text = lblLicensed.Text + " - temp";
}

```

Important Note!

When working with components, it's important to realize that the location of the license file when working with .NET needs to be different between design time and runtime.

Why?

Consider a solution where you have both the control and its container loaded and you are dropping an instance of a licensed control onto the container at design time. Visual Studio will load the control from the control's build directory (obj\debug or obj\release –for VB .Net developers – not the bin directory!). So that's where the license file has to be.

However, at runtime the component will be copied to the executable directory of the container and will run from there. The component will load – because it has a valid runtime key. However, the license component will not detect a valid license file unless you've copied it into that directory as well. But that's OK, and what you expect – because the licensing at runtime is based on the runtime key, which does not require a valid license in order to verify – so the component will successfully load without a DLSC file.

Potential Vulnerabilities

It is important to realize that any system that uses a runtime key suffers from the traditional vulnerability of any non-server based licensing. It depends on a secret, and thus is subject to attack.

The Desaware Licensing System does, however, add considerably more security over the LicFileProvider included with .NET (that really depends purely on trust, since all you need to do to break it is copy a text file). The security comes from the following:

- The Desaware Licensing System will not provide a runtime key unless you first perform a VerifyLicense operation which succeeds (with either a signed, or temporary certificate, both full and demo mode).
- You can improve security further by only returning the runtime key if the design time license is based on a signed certificate. If you do so, during the VerifyRuntimeKey function you can ignore any runtime keys that do not begin with the letter “S” (for signed)
- If your component is running under an unsigned license, the runtime key is generated using an encryption algorithm based on your application’s password.
- If your component is running under a signed license, the runtime key is provided by the server, and is itself digitally signed.
- Runtime keys are bound to the component name, but it is potentially possible for a malicious application to extract a runtime key and use it to enable your component. You can reduce the chances of this by customizing the runtime key to include additional information, perhaps binding the key to the container application. Runtime key generation and verification is reflected to your component, so such customization is not difficult.

Variations

- The .NET LicenseProvider system relies on the host (such as Visual Studio) to distinguish between runtime and design time. However, there are alternate approaches. For example: you might license your component so that it requires licensing to run under a debugger. In that case all you need to do is check for a debugger using the System.Debugger.IsAttached method. If the result is True, check the licensing component using VerifyLicense and do whatever you feel is appropriate (like throwing an error) if the component is unlicensed. This particular variation can be useful for embedded components in some situations.
- By default, a runtime key is available for both temporary and signed licenses, and to demo licenses. You can restrict this further in your GetLicenseKey implementation, for example: only returning runtime license keys if a signed license exists.
- The example shown here does not include installation. We have omitted this information as you have already seen how installation works.

Embedded Components

A significant limitation of the .NET licensing system is that when the host is in design mode, all components in the project are in design mode. There is no good mechanism for one component embedded inside another to recognize that it is embedded and be treated as if it were in run mode within the environment.

In other words – licensing of components that are embedded inside other components is problematic in the current .NET Framework.

The Desaware Licensing System does not address this scenario. Sorry folks – maybe for the next version.

Using Custom Data

The Installer sample application also demonstrates use of custom data to perform registration at the same time as an installation. This is just one possible use of custom data – it is appropriate for any situation where you wish to provide additional information during registration. There are, however, some things you must understand about the way custom data works before you design it into your system.

- Custom data is bound to a unique installation. What does this mean?
 - If someone reinstalls the software on the same system, using the same key or a different key, and provides new custom data, the existing data will be replaced.
 - If someone reinstalls the software on the same system, using the same key or a different key, and provides no custom data, the existing data will remain.
- Custom Data is replaced on a field by field bases. This means that if the licensing system is replacing existing custom data (due to a reinstall), each key will be replaced individually – and any keys that are not included on the new install will remain unchanged if they already exist in the database.
- Custom data is not encrypted in the licensing certificate. This is because the data is generated on the client system, and is therefore not assumed to be confidential. However, the custom data is encrypted (along with the entire certificate) when being registered on the server. If you wish custom data to be encrypted in the certificate, it is up to you to encrypt it before placing it in the certificate. In that case it will also be encrypted in the database.
- Custom data only goes in one direction – client to server. If you reinstall a software and do not provide custom data, any matching custom data on the server will not be added to the certificate when it is returned to the client.

Client Component Reference

The Desaware.MachineLicense component is the client component used for licensing.

ClientLicense Properties

This documentation will not include the methods and properties inherited from the parent Component class. That information can be found in the MS .NET Framework documentation.

AdditionalSystemIdentifiers	<pre>[VB] Property AdditionalSystemIdentifiers() As ISystemIdentifier() [C#] ISystemIdentifier [] AdditionalSystemIdentifiers</pre> <p>Refer to the section "Extending the Licensing System" for information on this property.</p>
ApplicationName	<pre>[VB] Property ApplicationName as String [C#] string ApplicationName</pre> <p>Set or retrieve the application name being licensed. You cannot override the application name set by the license resource (.resx) file.</p>
ApplicationPassword	<pre>[VB] Property Password as String [C#] string Password</pre> <p>Set the secret password for the application that is used for first pass authentication and for temporary certificates. You cannot override the application password set by the license resource file. If you do set this explicitly, you should take steps to obfuscate your assembly so it won't be obvious.</p>
CustomData	<pre>[VB] Property CustomData As StringDictionary [C#] StringDictionary CustomData</pre> <p>Set or retrieve the custom data for this certificate. The data in this StringDictionary is stored unencrypted in the certificate and is sent with the certificate to the licensing server upon activation. The CustomData "key" name has a maximum length of 64, the "value" data has a maximum length of 255. Only letters, numbers and the underscore ('_') are valid for the CustomData key name. Note that the StringDictionary object will first translate the key names to lower case before adding it to the string dictionary.</p>

	<p>The CustomData field does not change if you use the ClientLicense Object to verify a different license file (such as changing the ResourceName then calling VerifyLicense). It retains the previous string dictionary, you will need to set it to Nothing before calling VerifyLicense to verify a different license file.</p>
DeferredLicense	<p>[VB] ReadOnly Property DeferredLicensed As Boolean [C#] bool DeferredLicensed</p> <p>Indicates that a temporary license certificate was found. Valid only after calling the VerifyLicense method. Applies to both demo and full licenses.</p>
DemoDaysRemaining	<p>[VB] ReadOnly Property DemoDaysRemaining As Integer [C#] int DemoDaysRemaining</p> <p>The number of days remaining until a demo installation expires. Valid only after calling the VerifyLicense method and when DemoVersion is True.</p>
DemoExpiration	<p>[VB] Property DemoExpiration as Integer [C#] int DemoExpiration</p> <p>Set or retrieve the number of days a demo is valid. Use zero to disable support for demo licensing. Maximum value is 365. You cannot override the demo expiration set by the license resource file. This number will set the demo expiration for a temporary certificate. For signed certificates, you can use this property to set a shorter demo duration than specified in the application database, however you cannot exceed that value. If you set a higher value than exists in the database, the signed certificate will have its expiration date reduced according to the maximum value in the database.</p>
DemoRequireInternet	<p>[VB] Property DemoRequireInternet As Boolean [C#] bool DemoRequireInternet</p> <p>Applicable only for demo licenses. If this property is set when the VerifyLicense method is called, the ClientLicense object attempts to retrieve the current date from a date server on the internet instead of retrieving the current date from the system date. If the date could not be retrieved from the internet during</p>

	VerifyLicense, a DemoNoInternetDate result is returned.
DemoVersion	<p>[VB] ReadOnly Property DemoVersion As Boolean</p> <p>[C#] bool DemoVersion</p> <p>Indicates that a valid demo certificate was found (temporary or signed). Valid only after calling the VerifyLicense method.</p>
ExcludeDefaultSystemIdentifiers	<p>When set to True, the default system identifiers are not included in the list of system identifiers used to identify systems. Use this when you are using the AdditionalSystemIdentifiers property to define your own system identifiers, and do not wish to also include the default identifiers.</p> <p>The LicensedWebApp sample application demonstrates how you can use this feature to implement a license policy that is based purely on domain or IP address.</p>
ExpirationDate	<p>[VB] ReadOnly Property ExpirationDate() As Date</p> <p>[C#] DateTime ExpirationDate</p> <p>Returns the expiration date of this particular demo install from the license file. Only valid after calling the VerifyLicense function. If the license file is not a demo, then this date will be the same as the InstallationDate.</p>
InstallationDate	<p>[VB] ReadOnly Property InstallationDate() As Date</p> <p>[C#] DateTime InstallationDate</p> <p>Returns the installation date of this particular installation from the license file. Only valid after calling the VerifyLicense function.</p>
InstallationSerial	<p>[VB] ReadOnly Property InstallationSerial() As String</p> <p>[C#] string InstallationSerial</p> <p>Returns a unique GUID value describing this particular installation. Only valid for signed certificates, this matches the UniqueInstallGUID in the license server database (see the section on Database schema and contents). You might use this for serial numbers.</p>

InstallError	[VB] ReadOnly Property InstallError As InstallErrorResults [C#] InstallErrorResults InstallError Contains the most recent result of an Install or InstallDemo call. Also the result of a completed deferred installation (background operation).
InstallThread	[VB] ReadOnly Property InstallThread As Thread [C#] Thread InstallThread A reference to the background thread when doing a background install or deferred install (during verification). Can be used for synchronization purposes.
Licensed	[VB] ReadOnly Property Licensed As Boolean [C#] bool Licensed Indicates that a valid license certificate was found (temporary or signed). Valid only after calling the VerifyLicense method.
LicenseFilePath	[VB] Property LicenseFilePath as String [C#] string LicenseFilePath Set or retrieve the path of the license file. If not set, the license component will make a best guess of where the license file is (or should go). The LicenseFilePath is also valid after a successful call to VerifyLicense.
LicenseServers	[VB] Property LicenseServers As String() [C#] string [] LicenseServers Set or retrieve a list of URL's to license servers. The component will try all of them until one successfully licenses the component. You can override the default license server set by the license resource file using this property. NOTE: There is no automatic synchronization if you choose to use more than one license server. This is intended for advanced users who can figure out their own synchronization. Or for scenarios where the license servers are connected to the same database through the company's back end (such as in a web farm).
PublicKey	[VB] Property PublicKey as String [C#] string PublicKey

	<p>Set or retrieve the application name being licensed. You cannot override the public key set by the license resource file.</p>
ResourceName	<p>[VB] Property ResourceName As String [C#] string ResourceName</p> <p>Set or retrieve the name of the resource file containing the application name, application password, public key, demo expiration and default server URL.</p>
RuntimeKey	<p>[VB] ReadOnly Property RuntimeKey As String [C#] string RuntimeKey</p> <p>Returns a runtime key for use with the DlsLicenseProvider when licensing components. Only valid after calling the VerifyLicense method and when either DemoVersion or Licensed are True.</p>
ServerTimeout	<p>[VB] Property ServerTimeout as Integer [C#] int ServerTimeout</p> <p>The timeout in milliseconds of each server request.</p>
TimeServerSettings	<p>[VB] Public Property TimerServerSettings As TimeServerOptions [C#] TimeServerOptions TimeServerSettings</p> <p>The TimeServerOptions enumeration has the following values:</p> <p>0 – default – connect to Internet time servers only. 1 – Disable all connection to time servers.</p> <p>By default, the ClientLicense component always attempts to reach an external Internet Time server when performing a verification. The DemoRequireInternet property specifies whether a successful connection to an external time server is <i>required</i> when doing a demo validation – however the attempt is always made, even on an installation.</p> <p>Use this option to disable all access to external Internet time servers. These servers are accessed on port 13. While this option does reduce the security of demo licensing (as there is no way for the system to detect that the user has set back their system</p>

date), it does provide support for client scenarios that have specific security requirements.

In these cases the GetActivationServerTime function can be used as an alternate mechanism for obtaining an external date.

ClientLicense Methods

This documentation will not include the methods and properties inherited from the parent Component class. That information can be found in the MS .NET Framework documentation.

ClientLicense Constructor	<pre>[VB] Sub New(ByVal ApplicationName As String, ByVal Password As String, ByVal PublicKey As String) [C#] ClientLicense(string ApplicationName, string Password, string PublicKey)</pre> <p>This constructor allows you to initialize the ApplicationName, Password and PublicKey values.</p>
ClientLicense Constructor	<pre>[VB] Sub New(ByVal ResourceName As String) [C#] ClientLicense(string ResourceName)</pre> <p>This constructor accepts a resource file name. Do not include the .resx extension in the name. This is the preferred constructor.</p>
ClientLicense Constructor	<pre>[VB] Sub New(ByVal ResourceName As String, ByVal AssemblyContainingResource As Assembly) [C#] ClientLicense(string ResourceName, Assembly AssemblyContainingResource)</pre> <p>This constructor accepts a resource file name and a reference to the assembly containing the resource. Do not include the .resx extension in the name. This constructor is generally used when encapsulating the component into another..</p>
DeleteExistingCertificate	<pre>[VB] Public Sub DeleteExistingCertificate(ByVal IsolatedStorageName As String, ByVal StorageMode As SaveLicenseModes) [C#] public void DeleteExistingCertificate(String IsolatedStorageName, SaveLicenseModes StorageMode)</pre>

	<p>Use to delete a certificate from isolated storage.</p> <p>The StorageMode parameter should be set to one the three values: SaveToMachineIsolatedStorage, SaveToUserIsolatedStorage or SaveToMachineThenUserIsolatedStorage. Refer to the description of these values in the documentation for the SaveLicenseModes enumeration.</p> <p>The function does not verify the existing certificate and does not throw an exception or return a result in case of failure.</p>
GetActivationServerTime	<pre>[VB] Function GetActivationServerTime(Optional ByVal waittime As Integer = 150) As Date [C#] Date GetActivationServerTime(int waittime)</pre> <p>Use this function to retrieve the current date/time from your activation server. This can be useful in cases where port 13 is blocked. The function returns the value DateTime.MinValue if an error occurs.</p> <p>Keep in mind that using this function increases the load on your activation server. This can be significant if you have a large number of clients (remember, the Desaware license system was designed to operate in a largely disconnected manner and thus places a generally low load on the license server).</p> <p>You must have the version 1.3 or later license server installed to use this function.</p>
GetEncryptedServerData	<pre>[VB] Function GetEncryptedServerData (ByVal encryptkey As String, ByVal iv() As Byte, ByVal data As String) As String [C#] string GetEncryptedServerData (string encryptkey, byte [] iv, string data)</pre> <p>Decrypts the string specified in data. The encryptkey is a secret key used to encrypt and decrypt data. The iv byte array is an initialization vector also used for encrypting and decrypting data and must be set to exactly 16 elements. The same encryptkey and iv array must have been used to encrypt the data or this function will fail. This function is typically called to decrypt server data returned in the license file.</p> <p>This function returns Nothing if it cannot decrypt the data. This would most likely be caused by a</p>

	<p>mismatched key or iv, or the incorrect data to decrypt.</p> <p>NOTE: You can call the EncryptServerData function to encrypt any data string. But if you want to directly decrypt the byte array returned by the EncryptServerData function, you will need to call the Convert.ToBase64String function to convert the byte array into a base64 string before passing that string to this function. This is what the ClientLicense component does when writing the returned byte array from the ProvideServerData function.</p>
GetInternetTime	<pre>[VB] Function GetInternetTime (Optional ByVal WaitTime As Integer = 150) As Date [C#] DateTime GetInternetTime (int WaitTime)</pre> <p>Retrieves the current time from an internet time server. This function will attempt to connect to a list of internet time servers and retrieve the current time. The WaitTime parameter is an optional parameter specifying the maximum wait time in milliseconds for a connection to be established and data to be returned on each internet time server before trying the next.</p> <p>This function returns DateTime.MinValue if it cannot establish any connection with any of the internet time servers. If this occurs and an internet connection does exist, try lengthening the WaitTime and call this function again.</p>
GetResultDescription	<pre>[VB] Function GetResultDescription(ByVal ValidationResult As ValidationStatus) As String Function GetResultDescription(ByVal InstallationResult As InstallErrorResults) As String</pre> <pre>[C#] string GetResultDescription (ValidationStatus ValidationResult) string GetResultDescription (InstallErrorResults InstallationResult)</pre> <p>Returns a plain text description of the results indicated by the ValidationStatus and InstallErrorResults enumerations.</p>
InstallDemo	<pre>[VB] Function InstallDemo(ByVal InstallMode As InstallationModes) As InstallErrorResults [C#] InstallErrorResults InstallDemo (InstallationModes InstallMode)</pre> <p>This function installs a demo license. The</p>

	<p>DemoExpiration property must be non zero for this function to succeed. InstallMode is as described for the InstallLicense method.</p> <p>Refer to the InstallErrorResults enumeration for specifics on possible results.</p>
InstallLicense	<pre>[VB] Function InstallLicense(ByVal InstallationCode As String, ByVal InstallMode As InstallationModes) As InstallErrorResults [C#] InstallErrorResults InstallLicense (string InstallationCode, InstallationModes InstallMode)</pre> <p>This function installs a license. The InstallationCode is a properly formatted installation key. InstallMode can be one of the following:</p> <p>SyncActivationRequired – Performs a synchronous connection to the server to register the license. Licensing will fail if the server cannot be reached.</p> <p>SyncAllowDeferred – Performs a synchronous connection to the server to register the license. Will store a temporary license if the server cannot be reached.</p> <p>AsyncAllowDeferred – Performs an asynchronous connection to the server to register the license. Stores a temporary license before the asynchronous call, and replaces it with a signed one if the server connection is successful. Deletes the temporary license if the installation code is blocked or invalid</p> <p>Refer to the InstallErrorResults enumeration for specifics on possible results.</p>
StoreOrUpdateSignedCertificate	<pre>[VB] Public Function StoreOrUpdateSignedCertificate(ByVal IsolatedStorageName As String, ByVal StorageMode As SaveLicenseModes, ByVal doc as XmlDocument) As InstallErrorResults [C#] public InstallErrorResults StoreOrUpdateSignedCertificate(String IsolatedStorageName, SaveLicenseModes StorageMode, XmlDocument doc)</pre> <p>Use to store a signed certificate to isolated storage after it has been signed remotely. If the IsolatedStorageName parameter is null, the default isolated storage name will be used (the default is License\appname.dlsc where appname is your</p>

	<p>application name).</p> <p>The StorageMode parameter should be set to one the three values: SaveToMachineIsolatedStorage, SaveToUserIsolatedStorage or SaveToMachineThenUserIsolatedStorage. Refer to the description of these values in the documentation for the SaveLicenseModes enumeration.</p> <p>This function first verifies the signed license. Returns a result of InvalidCertificate or CertSaveError on error. Use the VerifyLicense function if necessary to determine the validation error that caused the InvalidCertificate error.</p>
VerifyAndRetrieveUnsignedCertificate	<pre>[VB] Public Function VerifyAndRetrieveUnsignedCertificate (ByVal IsolatedStorageName As String, ByVal StorageMode As SaveLicenseModes) As XmlDocument [C#] public XmlDocument VerifyAndRetrieveUnsignedCertificate (String IsolatedStorageName, SaveLicenseModes StorageMode)</pre> <p>Use to retrieve an unsigned certificate from isolated storage in order to sign remotely. If the IsolatedStorageName parameter is null, the default isolated storage name will be used (the default is License\appname.dlsc where appname is your application name).</p> <p>The StorageMode parameter should be set to one the three values: SaveToMachineIsolatedStorage, SaveToUserIsolatedStorage or SaveToMachineThenUserIsolatedStorage. Refer to the description of these values in the documentation for the SaveLicenseModes enumeration.</p> <p>This function first verifies the unsigned license and makes sure that it is unsigned. Returns the XmlDocument of the certificate on success (use the XmlDocument.Save method to save the file for transfer).</p> <p>If an error occurs, a MachineLicenseException exception will be thrown.</p>
VerifyInstallationCode	<pre>[VB] Function VerifyInstallationCode (ByVal InstallCode As String) As Boolean [C#] bool VerifyInstallationCode (string InstallCode)</pre>

	<p>Does a first pass verification of an installation code based on the application name and password. Used by the CodeEntryControl to determine if a password is valid. Does not perform verification against the license server.</p>
VerifyLicense	<pre>[VB] Public Function VerifyLicense(ByVal DeferredCheckInBackground As Boolean) As ValidationStatus [C#] public ValidationStatus VerifyLicense(bool DeferredCheckInBackground)</pre> <p>Verifies a license. If a temporary certificate is present, this function will try to register it with the licensing server. If DeferredCheckInBackground is True, the registration attempt will happen asynchronously, and this function will return a result based upon the temporary certificate.</p> <p>Refer to the ValidationStatus enumeration for specifics on possible results.</p>
VerifyLicense (override)	<pre>[VB] Public Function VerifyLicense(ByVal DeferredCheckInBackground As Boolean, ByVal IsolatedStorageName As String, ByVal StorageMode As SaveLicenseModes) As ValidationStatus [C#] public ValidationStatus VerifyLicense(bool DeferredCheckInBackground, String IsolatedStorageName, SaveLicenseModes StorageMode)</pre> <p>Verifies a license. If the IsolatedStorageName parameter is null, the default isolated storage name will be used (the default is License\appname.dllsc where appname is your application name).</p> <p>The StorageMode parameter should be set to one of the three values: SaveToMachineIsolatedStorage, SaveToUserIsolatedStorage or SaveToMachineThenUserIsolatedStorage. Refer to the description of these values in the documentation for the SaveLicenseModes enumeration.</p> <p>Refer to the ValidationStatus enumeration for specifics on possible results.</p>
VerifyLicense (override)	<pre>[VB] Public Function VerifyLicense(ByVal DeferredCheckInBackground As Boolean, ByVal SourceDocument As XmlDocument) As ValidationStatus</pre>

	<pre>[C#] public ValidationStatus VerifyLicense(bool DeferredCheckInBackground, XmlDocument SourceDocument)</pre> <p>Verifies a license. Use this override to verify a certificate saved using the CustomSave option. Before calling this function, load the externally stored data into an XmlDocument object.</p>
VerifyRuntimeKey	<pre>[VB] Function VerifyRuntimeKey(ByVal runtimekey As String) As Boolean [C#] bool VerifyRuntimeKey(string runtimekey)</pre> <p>Verifies a runtime key. Used with the DlsLicenseProvider class.</p>

ClientLicense Events

This documentation will not include the methods and properties inherited from the parent Component class. That information can be found in the MS .NET Framework documentation

DeferredInstallComplete	<pre>[VB] Event DeferredInstallComplete(ByVal Sender As Object, ByVal e As ClientLicenseEventArgs) [C#] DeferredInstallComplete (object Sender, ClientLicenseEventArgs e)</pre> <p>This event is raised on completion of a background (asynchronous) activation. The ClientLicenseEventArgs object contains one property of interest:</p> <p>InstallResults – An InstallErrorResults enumeration indicating the result of the attempted install.</p>
SaveLicense	<pre>[VB] Event SaveLicense(ByVal Sender As Object, ByVal e As SaveLicenseEventArgs) [C#] SaveLicense (object Sender, SaveLicenseEventArgs e)</pre> <p>This event is raised after a license has been issued but before it is stored. It allows the application the opportunity to modify the storage of the license certificate by modifying parameters of the SaveLicenseEventArgs object. The SaveLicenseEventArgs object has the following</p>

properties:

SaveLicenseMode – A SaveLicenseModes enumeration value. Set this value to the desired storage mode.

LicenseFilePath - When the SaveLicenseMode property is 0 (SaveToDisk), this property contains the current (default) path to the license file. You can modify this to save the file to a different location. Note that modifying this file does not modify the LicenseFilePath property of the licensing component – this is a temporary change just for this file.

IsolatedStorageName - When the SaveLicenseMode property is 2, 3 or 4 (SaveToMachineIsolatedStorage, SaveToUserIsolatedStorage or SaveToMachineThenUserIsolatedStorage), this property contains the file name that will be used to store the certificate in isolated storage. You can modify this file name if you wish.

XMLDoc - This property will always contain the XmlDocument object of the license certificate itself. When the SaveLicenseMode property is 1 (CustomSave) it is your responsibility to store the contents of this object in the manner you desire. Typically you can use the object's Save method to save the contents to a stream which can be a file or memory stream which can then be stored in the location you choose. You can also use the OuterXml property to obtain a string containing the XML for the document.

Do not modify the contents of this document.

Note that the property is a copy of the one used within the licensing component, so any attempts to modify the contents will have no effect in any mode other than CustomSave, and in that mode, changes to signed certificates will invalidate the certificate.

DoNotOverwriteExistingLicense - When the SaveLicenseMode property is 1 (CustomSave), examine this property to determine how to handle existing licenses. When True, if a license currently exists you should not overwrite it. This will typically happen when the new license is a demo certificate but the existing one is a full license (an attempt to install a demo on a fully licensed system).

KillRecentLicense	<pre>[VB] Event SaveLicense(ByVal Sender As Object, ByVal e As SaveLicenseEventArgs) [C#] SaveLicense (object Sender, SaveLicenseEventArgs e)</pre> <p>This event is raised when an existing license needs to be deleted. This will only occur when a temporary license has been just installed and activation fails due to an invalid or blocked code.</p> <p>This event includes a SaveLicenseEventArgs parameter as well that is a copy of the one actually used to create the certificate (during the previous SaveLicense event).</p> <p>If the SaveLicenseMode property is 0 (CustomSave), it is your responsibility to delete the previously created temporary certificate.</p> <p>Changes to the SaveLicenseEventArgs parameter during this event have no effect.</p>
--------------------------	---

SaveLicenseModes Enumeration

SaveToDisk = 0	This is the default value. During the SaveLicense event, setting this value indicates the license file should be saved to disk in the location specified by the LicenseFilePath property.
CustomSave = 1	During the SaveLicense event, setting this value indicates that you wish to store the certificate. When set, the licensing component will not store the license. In a typical application you might use this option to store the certificate in an external database. The certificate can be found in the XMLDoc parameter.
SaveToMachineIsolatedStorage = 2	During the SaveLicense event, setting this value indicates that the license file should be stored to isolated storage based on the application, domain and machine user (i.e., all users). The name of the isolated storage file is specified by the IsolatedStorageName property which is set by default to License\appname.dlsc but can be overridden (where appname is your application name). This is the

	preferred storage mode for web applications and services intended to be installed on hosted sites (where you are doing FTP deployment rather than using an installer package).
SaveToUserIsolatedStorage = 3	During the SaveLicense event, setting this value indicates that the license file should be stored to isolated storage based on the application, domain and current user. The name of the isolated storage file is specified by the IsolatedStorageName property which is set by default to License\appname.dlsc but can be overridden (where <i>appname</i> is your application name). This setting is rarely used, as in most cases it is desirable to license an application to all users rather than a single user.
SaveToMachineThenUserIsolatedStorage = 4	During the SaveLicense event, setting this value indicates that the license file should be stored to isolated storage based on the application, domain and machine user (i.e., all users), but if that fails to store it based on the current user. The name of the isolated storage file is specified by the IsolatedStorageName property which is set by default to License\appname.dlsc but can be overridden (where <i>appname</i> is your application name). This is an alternate storage mode for web applications and services intended to be installed on hosted sites (where you are doing FTP deployment rather than using an installer package), to handle cases where security does not permit access to the machine isolated storage. Because hosted sites typically run under a single user account (ASPNET or NETWORK_SYSTEM) user isolated storage is typically as effective as machine storage.

ValidationStatus Enumeration

The ValidationStatus enumeration defines the possible results of a call to the ClientLicense.VerifyLicense method.

Result	Definition
Success = 0	Successful verification of a signed demo or full license (use the ClientLicense DemoVersion or Licensed properties to determine which one).
TempSuccess = 1	Successful verification of temporary demo or full license (use the ClientLicense DemoVersion or Licensed properties to determine which one).
MissingOrIncorrectApplication = 2	The certificate is missing the application element, or it does not match the application name specified for this ClientLicense component.
SignatureVerificationError = 3	The digitally signed certificate is invalid or has been modified.
InvalidCertificate = 4	The format of the license certificate is invalid.
MissingInstallKey = 5	The Install key is missing on an application that does not support demonstration licenses.
IncorrectSystem = 6	The license certificate is not valid on this machine.
DemoExpired = 7	The demo license has expired.
DemoNoInternetDate = 8	An internet connection is required to verify the current date for this demo certificate but a connection could not be established with an internet date server.

InstallErrorResults Enumeration

The InstallErrorResults enumeration defines the possible results of a call to the ClientLicense InstallLicense or InstallDemo methods. It is also used in the DeferredInstallComplete event, and with the InstallError property.

Values under FatalError are informational only and represent successful installations.

Result	Definition
NoError = 0	Successful registration.

CodeReuseWarning = 1	Successful registration. The installation code has been used on a number of unique machines that exceeds the number specified by the Warning entry in the server database.
DeferServerError = 2	Successful registration of a temporary certificate. The server cannot be reached for registration. Can occur when SyncAllowDeferred is set as the installation mode.
AsyncRequestStarted = 3	Successful registration of a temporary certificate. A request to register on the server has started. Typically occurs when AsyncAllowDeferred is set as the installation mode.
NoLicenseServer = 4	Successful registration of a temporary certificate. No license servers can be reached.
DemoInstallOnPrevRegisteredSystem = 5	A demo installation has been successfully completed on a system that is registered in the server as already having a full installation. The component will not overwrite an existing full license file with a demo license file. See DemoInstallOnRegisteredSystem.
FatalError = 99	Marker – not returned as a result.
InvalidCode = 100	The installation code is invalid.
CodeReuseBlocked = 101	The installation code has been used on a number of unique machines that exceeds the number specified by the Block entry in the server database.
InvalidCertificate = 102	Registration failed. The certificate or public key is invalid.
ServerError = 103	Registration failed. The server is unable to process the request at this time.
NoSuchApplication = 104	Registration failed. The server does not have an entry for this application.
DemoInstallOnRegisteredSystem = 105	Registration failed. The current machine already has a full license. The component will not overwrite an existing full license file with a demo license file.
DemoExpired = 106	Registration failed. This system is already registered has having a demo installation for

	this application that has expired.
CertSaveError = 107	Registration failed. Unable to save the license certificate.
NoOperationCalled = 108	VerifyLicense, InstallLicense or InstallDemo have not yet been called.

The CodeEntryControl

The CodeEntryControl control is a simple control for entering and validating installation codes. This documentation will not include the methods and properties inherited from the parent Component class. That information can be found in the MS .NET Framework documentation

It is possible to paste values into the CodeEntry control, however the paste operation will only work once every 500ms. Additional attempts will be ignored.

CodeEntryControl Properties

License	[VB] Property License As ClientLicense [C#] ClientLicense License Set this property to an existing ClientLicense component to have the control perform immediate first pass verification of installation codes.
InstallCode	[VB] Property InstallCode() As String [C#] string InstallCode Use to retrieve a properly formatted installation code. Returns Nothing if the code is not 26 characters long and passes first pass validation. For security reasons, this property can only be set once every 500ms. Values set more frequently will be ignored.
ToolTip	[VB] Property ToolTip As String [C#] String ToolTip set the tooltip that appears over the control (including all of the text boxes in the control).

CodeEntryControl Methods

ClearText	[VB] Sub ClearText () [C#] void ClearText()
-----------	--

Clears all the text fields for the CodeEntryControl.
--

CodeEntryControl Events

LicenseCodeEntered	<div><div>[VB] Event LicenseCodeEntered(ByVal Sender As Object, ByVal e As CodeEntryEventArgs)</div><div>[C#] event LicenseCodeEntered (object Sender, CodeEntryEventArgs e)</div><div>This event is raised any time the known first pass validation status of the contents of the control changes. The CodeEntryEventArgs parameter contains two important parameters:</div><div>ValidationChecked indicates that the results were verified against the ClientLicense component specified by the License property. It will always be True if the License property is set.</div><div>IsValid indicates that the current contents of the control represents a valid license key based on a first pass verification (does not include verification against the server's list of allocated installation codes).</div></div>
--------------------	--

Additional Information

The following additional information will help you make full use of the MachineLicense component.

Code Access Security

The Client licensing component requires full trust in order to work correctly. However, once you have installed it on a system (for example: in the GAC), it can be used to implement licensing for partially trusted components.

How the Component is Licensed

The MachineLicense component is itself licensed on a per machine/server basis. You can distribute it freely with your applications. Note, however, that any attempt to perform an Install operation using the component while running under a debugger requires the component to be licensed on that system.

Code Reuse Blocking

In order to reduce the chance of performance bottlenecks, the licensing server does not lock access to the installation key database between the time it extracts verification information, and updates the database with a new registration. In other words, there is a finite possibility that two different systems using the same installation code will both be allowed to install even if this would cause the number of uses for the key to be exceeded.

For this reason, we do not absolutely guarantee that the code reuse blocking value will be enforced. However, for this problem to occur two requests on the same installation key would have to arrive and be processed virtually simultaneously – an event unlikely to occur in practice. When this problem does occur, the next request will be correctly blocked.

The WebCodeEntry Control

The WebCodeEntryControl is a new ASP .NET server control that is designed for entering installation codes. It supports many of the same features as the existing CodeEntryControl including automatic navigation between the text entry boxes, paste operations, and AJAX based code validation with the ability to enable a secondary control such as a command button when a code is valid.

Using the Desaware.WebCodeEntry control

The Desaware.WebCodeEntry control is contained in the assembly Desaware.WebCodeEntry.dll. To use the control, add a reference to the assembly from your web application or service.

You can add ASP .NET web pages to web service projects.

After adding a reference, the component should appear in your toolbar and can be dragged onto an ASP .NET web form.

The advanced functionality of the control will only work if JavaScript is enabled on the browser.

When a form containing the control is submitted, the InstallCode property will contain the installation code that was entered.

Desaware.WebCodeEntry control properties

ClientLicense	Set this property to an instance of a ClientLicense object if you want to enable automatic first-pass validation of entered codes. If this property is not set, any 26 character text string will be considered valid and returned in the InstallCode property.
InstallCode	Readonly property. Will return a properly formatted installation code if a valid code was entered. If a ClientLicense component was assigned to the ClientLicense property, this property will return a value only if it passes first pass validation of the code value. Otherwise, any 26 character code will be considered valid.
EnableControl	String property containing the ID attribute for another control on the form. The control (typically a command button, but can be any control with a disabled property) will be enabled when the control is empty, or has a valid installation code.

Refer to the LicensedWebApp sample application for an example of using the control.

Client Side Support

The WebCodeEntry control provides limited client side code support in addition to the server side features. You can add a reference to the method “LicenseCodeEntered” to the control in

order to receive notification as to when a valid license code has been entered. The following code illustrates how you can attach a notification handler, then retrieve a license code and store it in a text control when the notification arrives:

```
<script language="javascript">
    function codeentered(codevalue)
    {
        document.getElementById("TextBox1").value = codevalue;
    }

    WebCodeEntryControl1.LicenseCodeEntered = codeentered;
</script>
```

Web Service Reference

The first thing you should know about the web service reference is, you may never need it. This is designed for people who wish to create their own management interface instead of using the License Manager application we provide, or to extend the License Manager.

Connecting to the Web Service

In order for .NET to automatically create a proxy to a web service, it's wsdl information must be available. By default, we disable this for security reasons. In order to reenale this, comment out or delete the <wsdlHelpGenerator> element in the webServices section of the server's Web.Config file as described here.

```
<!-- If you are adding your own web reference (to create your own management
extensions you must comment out the following lines_in order to obtain the
WSDL needed to build the proxy
-->
<webServices>
    <wsdlHelpGenerator href="helppage.htm" />
</webServices>
```

You can re-enable security after the proxy is created.

Web Service Methods

Because we only expect advanced users to connect directly to the service, the methods of the service were designed for efficiency (to minimize round-trips), and not necessarily for ease of use.

Management Web Service

GetApplicationResX	<pre>[VB] Function GetApplicationResX(ByVal ApplicationName As String, ByVal ServerUrl As String) As Byte() [C#] byte [] GetApplicationResX(string ApplicationName, string ServerUrl)</pre> <p>This method is used to retrieve a resource file for an application (the .resx file used by the client license component). The ServerUrl is the URL you are using to access the server. It will be embedded in the resource.</p> <p>You can save the file by creating a FileStream object to a .resx file and writing it as follows:</p> <pre>fw.Write(rx, 0, rx.Length)</pre>
GetApplicationList	<pre>[VB] Function GetApplicationList() As DataSet [C#] DataSet GetApplicationList()</pre>

	<p>Returns a DataSet containing a list of all applications defined on the server. Contains one table on success. Columns are:</p> <p>ApplicationName: The short name of the application.</p> <p>ApplicationDescription: The description of the application.</p>
CreateNewApplication	<pre>[VB] Sub CreateNewApplication(ByVal ApplicationName As String, ByVal Description As String, ByVal warningcount As Integer, ByVal blockcount As Integer, ByVal demoexpiration As Integer, ByVal password As String) [C#] CreateNewApplication(string ApplicationName, string Description, int warningcount, int blockcount, int demoexpiration, string password)</pre> <p>Creates a new application on the current server. Refer to the database schema for descriptions of each parameter (which corresponds to columns in the Application table). The password field may be null, in which case the service will create a random password. We recommend that approach.</p>
GetApplicationInfo	<pre>[VB] Function GetApplicationInfo(ByVal ApplicationName As String) As DataSet [C#] DataSet GetApplicationInfo(string ApplicationName)</pre> <p>Retrieves information about an application. The DataSet contains a single table and a single row on success. Refer to the Application table schema for descriptions of each column. All of the columns in the Application table are returned, except that only the public key part of the signature key is returned.</p>
CreateInstallationCodes	<pre>[VB] Function CreateInstallationCodes(ByVal ApplicationName As String, ByVal Count As Integer) As String() [C#] string [] CreateInstallationCodes(string ApplicationName, int Count)</pre> <p>Creates new installation codes for the specified application and returns them in a string array. Specify the number of codes in the Count parameter.</p>
GetExistingInstallCodes	<pre>[VB] Function GetExistingInstallCodes(ByVal appname As String, ByVal filter As String) As DataSet [C#] DataSet GetExistingInstallCodes(string appname, string filter)</pre>

	<p>Retrieves a set of existing installation codes for an application. The filter parameter can contain up to 5 letters that start the installation code you're looking for, or * to retrieve all codes for the application.</p> <p>Refer to the InstallationCodes table schema for descriptions of each column. All of the columns in the InstallationCodes table are returned. An additional column "InstallationCode" is included in the data set that contains the 26 character installation code.</p>
SetInstallCodeProperty	<pre>[VB] Sub SetInstallCodeProperty(ByVal InstallGuid As String, ByVal PropertyName As String, ByVal PropertyValue As Object) [C#] SetInstallCodeProperty(string InstallGuid, string PropertyName, object PropertyValue)</pre> <p>Use this method to set fields in the InstallationCodes table. The InstallGuid value can be obtained using the GetExistingInstallCodes method. The PropertyName is the column to change. The PropertyValue is the new value to enter.</p> <p>This method is typically used to set the Warning Count, Block Count or user defined data for the installation code.</p> <p>The CodeGUID, ApplicationName and CodeStart fields cannot be modified using this function.</p>
GetInstallKeyInfo	<pre>[VB] Function GetInstallKeyInfo(ByVal Appname As String, ByVal InstallCode As String) As DataSet [C#] DataSet GetInstallKeyInfo(string Appname, string InstallCode)</pre> <p>This method retrieves detailed information about the use of the specified installation code. The Appname represents the application name. The InstallCode is the 26 character installation code.</p> <p>The DataSet contains one table. Each row represent an entry with the following information:</p> <ul style="list-style-type: none"> • InstallationCodes.WarningCount • InstallationCodes.BlockCount • InstallationCodes.UserDefined • All fields from the UniqueInstalls table. • CustomData.CustomName

- CustomData.CustomValue

Each unique installation that uses the code is included in the results, with one line for each element of custom data associated with that installation.

For you SQL mavens, here's the query we use:

```
SELECT InstallationCodes.WarningCount,
InstallationCodes.BlockCount,
InstallationCodes.UserDefined,      "UniqueInstalls.*",
CustomData.CustomName, CustomData.CustomValue
FROM InstallationCodes INNER JOIN (UniqueInstalls
LEFT JOIN CustomData ON
UniqueInstalls.UniqueInstallGUID =
CustomData.UniqueInstallGUID) ON
InstallationCodes.CodeGUID = UniqueInstalls.CodeGUID
WHERE (((InstallationCodes.CodeGUID)='installcode'))
```

GetInstallKeyInfo2

```
[VB] Function GetInstallKeyInfo2 (ByVal Appname
As String, ByVal InstallCode() As String) As
DataSet
[C#] DataSet GetInstallKeyInfo (string Appname,
string [] InstallCode)
```

Similar to GetInstallKeyInfo, this method retrieves detailed information about the use of the specified installation codes. The Appname represents the application name. The InstallCode array contains a list of the 26 character installation code to retrieve detailed information for.

The DataSet contains one table. Each row represent an entry with the following information:

- InstallationCodes.CodeGUID – the text returned by this field is not the same as what is saved in the DB. The actual installation key code is returned in this field rather than a GUID representation of it.
- InstallationCodes.WarningCount
- InstallationCodes.BlockCount
- InstallationCodes.UserDefined
- All fields from the UniqueInstalls table.
- CustomData.CustomName
- CustomData.CustomValue

Each unique installation that uses the code is included in the

	<p>results, with one line for each element of custom data associated with that installation.</p> <p>Unlike the GetInstallKeyInfo function, this query will result in information for a key even if it has not been used yet in an installation.</p> <p>For you SQL mavens, here's the query we use (the WHERE term is extended to include all install codes specified in the function parameter):</p> <pre>"SELECT InstallationCodes.CodeGUID, InstallationCodes.WarningCount, InstallationCodes.BlockCount, InstallationCodes.UserDefined, UniqueInstalls.UniqueInstallGUID, UniqueInstalls.DemoExpiration, UniqueInstalls.InstallationDate, CustomData.CustomName, CustomData.CustomValue FROM InstallationCodes LEFT JOIN (UniqueInstalls LEFT JOIN CustomData ON UniqueInstalls.UniqueInstallGUID = CustomData.UniqueInstallGUID) ON InstallationCodes.CodeGUID = UniqueInstalls.CodeGUID WHERE (((InstallationCodes.CodeGUID)='installcode'))"</pre>
DevUtilities	<pre>[VB] Function DevUtilities(ByVal op As Integer) As Object [C#] object DevUtilities(int op)</pre> <p>Call this method with op = 0 to verify or create the necessary tables on a database. You may need to use this if the installation program is unable to initialize the tables.</p> <p>Call this method with op = 1 to determine if the licensing server is licensed for single application or full functionality. Returns the string "single" for single application, "full" for full license, and "error" if there is an error (typically a security violation – use the Diagnostics function to obtain details).</p>
SignAndRegisterDLSC	<pre>[VB] Function SignAndRegisterDLSC (ByVal AppName As String, ByVal datastream As String, ByVal BackupInstallCode As String, ByRef SignResults As SigningResults) As String [C#] string SignAndRegisterDLSC(string AppName, string datastream, string BackupInstallCode, SigningResults SignResults)</pre> <p>Use to sign a temporary DLSC file, using system identifiers in the existing file (rather than those on the current system). To load the Datastream string, load an existing temporary</p>

	<p>DLSC file using XmlDocument.Load, then use XmlDocument.Save to save it into a string (using a StringWriter object).</p> <p>If the certificate is not a demo certificate, the installation code within the certificate is used. Otherwise the BackupInstallCode (if present) is used. The results are of type SigningResults. Refer to the InstallErrorResults enumeration description for the meanings of the SigningResults enumeration values (note the numeric values differ between this enumeration and the InstallErrorResults enumeration).</p> <p>The return value is a string which can be loaded into an XmlDocument using the XmlDocument.LoadFrom method, then saved to a file using XmlDocument.Save.</p>
Test	<pre>[VB] Function Test() As String</pre> <pre>[C#] string Test()</pre> <p>Simple test to check whether or not you have access to the licensing server. Returns empty string on success, returns "error" if you do not have access to the licensing server.</p>
Diagnostics	<pre>[VB] Function Diagnostics () As String</pre> <pre>[C#] string Diagnostics ()</pre> <p>Performs advanced diagnostic checks. Returns expanded trace diagnostic errors. Note that the web.config file now has diagnostics turned on by default. You need to disable this when done testing (look for the key <add key="enablediagnostics" value = "true" /> and set the value to "false").</p>

Activator Web Service

GetServerDate	<pre>[VB] Function GetServerDate() As Date()</pre> <pre>[C#] DateTime GetServerDate()</pre> <p>This method is used to retrieve the current system date on the license server.</p> <p>If there is an active internet connection on the system your application is running on, you can use this method to retrieve the system date from another source when testing for any types of date expiration. To ease the load on your server, we recommend using the GetInternetTime function from the ClientLicense object as an alternative.</p>
---------------	---

Extending the Licensing System

There are four areas where the Desaware Licensing System can be extended¹:

System Identifiers	Generating data that uniquely identifies a system.
System Matching Algorithms	The algorithms that decide if two systems are identical.
Post Installation Actions	Performing actions after an installation.
Adding data to license certificates	Add server data to the client's license certificate.

System Identifiers

A System Identifier is a value that has a high probability of uniquely identifying a system. Some system identifiers provide an extremely high probability of being unique – the MAC value of an Internet adapter is universally unique for example. Others, like the server name, are less likely to be unique. The actual determination of whether two systems match depends on the system matching algorithm in use (which will be described later).

Each system identifier consists of a name and value. The name can be any descriptive name containing letters and numbers of up to 50 characters (no spaces).

The value should be a Base64 representation of a 256 bit hash of the value. Using a hash in this manner allows you to uniquely identify a system on the server, without actually sending any information that might violate the client's privacy.

The Samples\Advanced\SystemIds project demonstrates how to use custom system identifiers.

Define a class that implements the `ISystemIdentifiers` interface. This interface contains two properties. The `Name` property should return the name of the identifier. The `Values` property returns an array of system identifiers values corresponding to that name. In most cases this will be one value, but in others there may be multiple values (i.e. a system that has more than one network card would return the address of each card in the system).

[VB]

```
Imports System.Security.Cryptography
Public Class NewSystemId
    Implements Desaware.MachineLicense.ISystemIdentifier

    Shared Sub New()
        AppDomain.CurrentDomain.SetPrincipalPolicy( _
            System.Security.Principal.PrincipalPolicy.WindowsPrincipal)
    End Sub

    Public ReadOnly Property Name() As String _
    Implements Desaware.MachineLicense.ISystemIdentifier.Name
        Get
```

¹ You can also purchase a site/source license that allows you to customize any part of the licensing system.

```

        Return "User"
    End Get
End Property

Public ReadOnly Property Values() As String() _
Implements Desaware.MachineLicense.ISystemIdentifier.Values
    Get
        Dim UniqueString As String = _
        Principal.WindowsIdentity.GetCurrent().Name
        Return New String() {GetHash256OfString(UniqueString)}
    End Get
End Property

Private Function GetHash256OfString(ByVal source As String) _
As String
    Dim sh As New SHA256Managed()
    Dim b() As Byte
    sh.ComputeHash(System.Text.ASCIIEncoding.ASCII.GetBytes( _
    source))
    b = sh.Hash
    sh.Clear()
    Return (Convert.ToBase64String(b))
End Function

End Class

```

[C#]

```

public class NewSystemId:ISystemIdentifier
{
    static NewSystemId()
    {
        AppDomain.CurrentDomain.SetPrincipalPolicy(
            System.Security.Principal.PrincipalPolicy.
            WindowsPrincipal);
    }

    public string Name
    {
        get
        {
            return "User";
        }
    }

    public string[] Values
    {
        get
        {
            string UniqueString = System.Security.Principal.
                WindowsIdentity.GetCurrent().Name;
            return new String[] {GetHash256OfString(UniqueString)};
        }
    }

    private string GetHash256OfString(string source)

```

```

    {
        SHA256Managed sh = new SHA256Managed() ;
        Byte[] b;
        sh.ComputeHash(System.Text.ASCIIEncoding.ASCII.
            GetBytes(source));
        b = sh.Hash;
        sh.Clear();
        return (Convert.ToBase64String(b));
    }
}

```

The SystemIds project is substantially the same as the High Security project described earlier. To use the new system identifier, the following code is added to the form's Load event.

[VB]

```

Dim NewIdentifiers(0) As ISystemIdentifier
NewIdentifiers(0) = New NewSystemId()

ClientLicense1.AdditionalSystemIdentifiers = NewIdentifiers

```

[C#]

```

ISystemIdentifier[] NewIdentifiers = new ISystemIdentifier[1];
NewIdentifiers[0] = new NewSystemId();

ClientLicense1.AdditionalSystemIdentifiers = NewIdentifiers;

```

This code should be executed before a verification takes place.

System Matching Algorithms

The System Matching algorithm determines whether two systems are identical for the purpose of licensing. Each license certificate contains the system identifiers for the system on which it is installed. The System Matching algorithm is executed both on the client and server. On the client, it verifies that the certificate belongs to the system on which it is running. On the server, it determines whether an installation code is being used on a new system, or upon one which it has already been used.

The System Matching algorithm actually implements two separate comparisons – Installation match and Demo match. It is important to understand both.

Installation Match Algorithm

An installation match test answers the question: “Are these two systems possibly the same?”. This algorithm allows you to take into account the possibility of hardware or configuration changes on a system by making the matching less restrictive. On the client, if a certificate passes this test, it is considered valid for this system. On the server, if an installation request comes in for a particular installation code, and another use of that code is found with system identifiers

that match using this algorithm, the server assumes it is a reinstall of the code on an existing system, and not a new use of the installation code.

The key thing to remember about the Installation Match algorithm is that it should be the most forgiving. It errs in the side of finding a match.

The default Installation Match algorithm checks all system identifiers and considers two systems a match if any one identifier matches.

When an installation match occurs, a demo match is also performed. If the demo match succeeds, the system assumes a higher degree of confidence that the two systems match, so any custom data provided is updated in the database for this installation. If the demo match fails, the database is not updated with custom data, the installation is not counted against the number of allowed installations for the code, and the system identifier list for the installation is not updated. In other words, when performing a match, later installations are always compared against the first installation – not any subsequent installations that are allowed

Demo Match Algorithm

The demo match test answers the question: “Are you absolutely certain these two systems are the same?”. This test is not used on the client.

On the server, this test is used in two places:

- During a demo installation, if an exact match is found for these system identifiers, the installation found is checked to see if it had previously had a demo installed that has expired. If so, a demo expiration error is returned. This is the primary use of this algorithm – hence the name.
- During an full installation, if an exact match is found for the current installation code, the server assumes the two systems are identical and updates any existing custom data on the server

The default Demo Match algorithm checks all system identifiers and considers two systems a match if all of the identifiers match.

Defining a Custom System Match Algorithm

You might want to define your own system match algorithm in order to strengthen security or adapt it to your own needs. It is important that you implement the same algorithm on both the client and server (though, as you’ll note, the method with which you install matching algorithms differs between the two).

To implement a System Match Algorithm, you must first add a reference to the assembly `Desaware.Dls.Interfaces.dll`. This assembly contains a number of interfaces and class definitions that are common to both the client and server, including those used for System Match information.

You then define a class that implements the `Desaware.Dls.ISystemMatch` interface.

In the example that follows, also in the `SystemIds` project, a very strict form of security is implemented which is based upon both the User system identifier defined earlier, and the network card address. Both the Demo Match and Installation Match algorithms are the same –

basically saying that for two systems to match, the user name and at least one network card address must match. If either does not match, the systems will be considered different.

[VB]

```
Imports Desaware.Dls
Public Class NewSystemMatch
    Implements Desaware.Dls.ISystemMatch

    Public Function CheckDemoMatch(ByVal Identifiers() As _
        SystemIdentifierInfo, ByVal ExistingEntries() As _
        SystemIdentifierInfo) As Boolean _
        Implements ISystemMatch.CheckDemoMatch
        Dim thisidentifier, testidentifier As SystemIdentifierInfo
        Dim nicmacvalid, uservalid As Boolean
        For Each thisidentifier In Identifiers
            If thisidentifier.Name = "User" Then
                For Each testidentifier In ExistingEntries
                    If thisidentifier.Value = testidentifier.Value Then
                        uservalid = True
                        Exit For
                    End If
                Next
            End If
            If thisidentifier.Name = "NICMAC" Then
                For Each testidentifier In ExistingEntries
                    If thisidentifier.Value = testidentifier.Value Then
                        nicmacvalid = True
                        Exit For
                    End If
                Next
            End If
        Next
        If uservalid AndAlso nicmacvalid Then Return True
    End Function

    Public Function CheckInstallationMatch(ByVal Identifiers() _
        As SystemIdentifierInfo, ByVal ExistingEntries() As _
        SystemIdentifierInfo) As Boolean Implements _
        ISystemMatch.CheckInstallationMatch
        Return CheckDemoMatch(Identifiers, ExistingEntries)
    End Function
End Class
```

[C#]

```
public class NewSystemMatch:ISystemMatch
{
    public bool CheckDemoMatch(Desaware.Dls.SystemIdentifierInfo[]
        Identifiers, Desaware.Dls.SystemIdentifierInfo[] ExistingEntries)
    {
        bool nicmacvalid = false, uservalid = false;
        foreach(SystemIdentifierInfo thisidentifier in Identifiers)
        {
            if (thisidentifier.Name == "User")
```

```

        {
            foreach(SystemIdentifierInfo testidentifier in
                ExistingEntries)
            {
                if (thisidentifier.Value == testidentifier.Value)
                {
                    uservalid = true;
                    break;
                }
            }
        }
        if (thisidentifier.Name == "NICMAC")
        {
            foreach(SystemIdentifierInfo testidentifier in
                ExistingEntries)
            {
                if (thisidentifier.Value == testidentifier.Value)
                {
                    nicmacvalid = true;
                    break;
                }
            }
        }
        if (uservalid && nicmacvalid ) return true;
        return false;
    }

    public bool CheckInstallationMatch(
        Desaware.Dls.SystemIdentifierInfo[] Identifiers,
        Desaware.Dls.SystemIdentifierInfo[] ExistingEntries)
    {
        return CheckDemoMatch(Identifiers, ExistingEntries);
    }
}

```

Client Side Match Algorithm Registration

On the client side, you can use the RegisterMatchProcess method to register your System Match algorithm.

[VB]

```

SystemMatch.RegisterMatchProcess( _
ClientLicense1.ApplicationName, New NewSystemMatch())
ClientLicense1.LicenseFilePath = _
Environment.GetFolderPath(Environment.SpecialFolder.Personal)

```

[C#]

```

SystemMatch.RegisterMatchProcess(
ClientLicense1.ApplicationName, new NewSystemMatch());
ClientLicense1.LicenseFilePath =
Environment.GetFolderPath(Environment.SpecialFolder.Personal);

```


Important Note:

As mentioned earlier, the Desaware Licensing System is designed for per machine/server licensing. This example demonstrates a way to extend it to perform user licensing. Basically you're tricking the licensing system into seeing one machine as a different machine for each user. If you take this approach, remember to use the `LicenseFilePath` property of the client license component to place the license certificate in a directory associated with each individual user. The default location for license certificates is the project executable directory. If you leave the default, each time a user does an install, the existing certificate for the previous user will be overwritten!

Server Side Match Algorithm Registration

Server side matching algorithms are placed in a DLL assembly. You can have more than one matching algorithm in each assembly.

In this example, the code from the `NewSystemMatch.vb` file from the previous example is compiled into a separate DLL.

This DLL is placed in the bin directory of the licensing server.

Next, you must create or modify the configuration named `Desaware.LicenseServer.Dll.Config` file located in the same directory. Add an entry under the configuration/`SystemMatchAlgorithms` section as shown here:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <SystemMatchAlgorithms>
    <add application="Test"
      value="ServerMatch.dll,ServerMatch.NewSystemMatch" />
  </SystemMatchAlgorithms>
</configuration>
```

The `application` attribute is the name of the application for this algorithm. Use the application name "default" (case sensitive) to set the default algorithm for all applications.

The `value` attribute contains the name of the DLL containing your match algorithms, in addition to the object name that implements the `ISystemMatch` interface described earlier.

Remember that this DLL executes in the context of the ASP. Net user (unless you have specified a different account). If an exception is raised in your code, the licensing server will respond with a Server Error to the current operation.

The Desaware Licensing System includes a `SystemIDViewer` utility that displays the `SystemID` values for the default systemid fields (machine name, drive volume id, and network adapter) the Licensing System verifies.

Creating Post Installation Actions

Not supported on the single application version of the licensing system.

You can receive notification from the license server whenever a new entry is created in the database.

To implement a post installation action, you must first add a reference to the assembly `Desaware.Dls.Interfaces.dll`. This assembly contains the `ILicenseDataEntry` interface used for post installation actions.

You then define a class that implements the `Desaware.Dls.ILicenseDataEntry` interface as shown here:

[VB]

```
Imports Desaware.Dls
Public Class PostAction
    Implements Desaware.Dls.ILicenseDataEntry

    Public Sub LicenseDataEntered(ByVal UniqueInstallGUID As String, _
        ByVal EntryType As DataEntryTypes) Implements _
        Desaware.Dls.ILicenseDataEntry.LicenseDataEntered
        Dim entry As String
        Select Case EntryType
            Case DataEntryTypes.NewDemo
                entry = "New demo was created"
            Case DataEntryTypes.NewLicense
                entry = "New license was created"
            Case DataEntryTypes.UpdatedCustomData
                entry = "Custom data was updated"
        End Select

        Dim outputfile As New IO.StreamWriter("c:\temp\installog.txt")
        outputfile.WriteLine(entry & " - GUID: " & UniqueInstallGUID)
        outputfile.Close()
    End Sub
End Class
```

[C#]

```
public class PostAction: ILicenseDataEntry
{
    public PostAction()
    {
    }

    public void LicenseDataEntered(string UniqueInstallGUID,
        Desaware.Dls.DataEntryTypes EntryType)
    {
        string entry=null;
        switch(EntryType)
        {
            case DataEntryTypes.NewDemo:
                entry = "New demo was created";
                break;
            case DataEntryTypes.NewLicense:
                entry = "New license was created";
                break;
            case DataEntryTypes.UpdatedCustomData:
```

```

        entry = "Custom data was updated";
        break;
    }
    StreamWriter outputfile = new StreamWriter(
        "c:\\temp\\installog.txt");
    outputfile.WriteLine(entry + " - GUID: " + UniqueInstallGUID);
    outputfile.Close();
}
}

```

In this example, entries are simply logged. A more sophisticated application could access the license server database to obtain the detailed information for this UniqueInstallGUID (a GUID that identifies this installation) and perform any additional tasks. For example: you could extract the CustomData sent with the certificate, and perform online registration tasks. The EntryType indicates if the entry is a new license, a new demo license, or an update of custom data. A new license or demo license EntryType may also include an update of custom data.

You must build this class into a DLL and place it in the bin directory of the licensing server.

Next, you must create or modify the configuration named Desaware.LicenseServer.Dll.Config file located in the same directory. Add an entry under the configuration/LicenseDataNotify section as shown here:

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <LicenseDataNotify>
    <add application="Test"
      value="ServerMatch.dll,ServerMatch.PostAction" />
  </LicenseDataNotify>
</configuration>

```

Exceptions caused by your DLL will be ignored.

Adding data to license certificates

Not supported on the single application version of the licensing system.

You can add server generated data to a license certificate. This can be used to add information that your application can use to fine tune licensing (for example: control the number of clients that can access a service).

The server generated data consists of a byte array that you specify. The byte array is stored in the .DLSC file under the ServerData tag. The data format is XML encoded base64 data.

The server generated data is not encrypted by default, however it is digitally signed and thus cannot be modified.

Note: The following is a preliminary implementation of server data. A more advanced version (or at least one easier to use) is planned for the next version. However, this implementation will continue to be supported in future versions as well.

To add server generated data to a license certificate, you must create a class that contains the public method “ProvideServerData” with the following declaration:

```
[VB]
Public Function ProvideServerData(ByVal appname As String, _
ByVal UniqueGuid As String, ByVal InstallationCode As String, _
ByVal CustomData As Specialized.StringDictionary) As Byte()

[C#]
public byte[] ProvideServerData(string appname, string UniqueGuid, string
InstallationCode, System.Specialized.StringDictionary CustomData)
```

Based on the information provided, you can return any byte data you wish.

At the time this function is called, your licensing database has already been updated with the license information. The InstallationCode parameter will be null for demo installations. The licensing system does not store the server data you provide – it just passes it on to the client.

If you wish to return a string, use the System.Text.UnicodeEncoding class to convert the string into a byte array.

You can also define an object and serialize it using the Binary or Soap serialization mechanism.

Here’s an example from the ServerMatch sample project in which a string is embedded in the .DLSC file.

[VB]

```
Public Function ProvideServerData(ByVal appname As String, _
ByVal UniqueGuid As String, ByVal InstallationCode As String, _
ByVal CustomData As Specialized.StringDictionary) As Byte()
    Dim result As New Text.StringBuilder()
    result.Append("Installation date: " & Now())
    result.Append(ControlChars.CrLf)
    result.Append("Number of licenses: 50")
    result.Append(ControlChars.CrLf)
    result.Append("Override demo expiration: 1/1/2005")
    result.Append(ControlChars.CrLf)
    result.Append("License expiration date: 6/1/2006")
    result.Append(ControlChars.CrLf)
    Dim uc As New Text.UnicodeEncoding()
    Return uc.GetBytes(result.ToString)
End Function
```

[C#]

```
public byte[] ProvideServerData(string appname, string UniqueGuid, string
InstallationCode, System.Specialized.StringDictionary CustomData)
{
    System.Text.StringBuilder result =
    new System.Text.StringBuilder();
    result.Append("Installation date: " +
    System.DateTime.Now.ToString() );
    result.Append("\n");
    result.Append("Number of licenses: 50\n");
    result.Append("Override demo expiration: 1/1/2005\n");
    result.Append("License expiration date: 6/1/2006\n");
    // Why a byte array? It provides the most flexibility
```

```

        // (say, if you're doing encryption),
        // and supports binary serialization of objects
        System.Text.UnicodeEncoding uc = new
        System.Text.UnicodeEncoding();
        return uc.GetBytes(result.ToString());
    }

```

You must build the class that provides server data into a DLL and place it in the bin directory of the licensing server.

Next, you must create or modify the configuration file named `Desaware.LicenseServernn.Dll.Config` located in the same directory (*nn* is 11 or 20 depending on framework version). Add an entry under the configuration/ProvideServerData section as shown here:

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <ProvideServerData>
    <add application="Test"
      value="ServerMatch.dll,ServerMatch.ServerData" />
  </ProvideServerData>
</configuration>

```

The *application* string in the ProvideServerData section should be set to the name of your Application as defined by the Licensing System. The *value* string format should be “<assembly file name>,<Namespace.ClassName_Containing_ProvideServerData>”. Each application for which you want to provide server data must have its own <add> key, and each application may have more than one of these keys.

Unhandled exceptions caused by your DLL will be ignored.

If the file or object specified in the configuration file does not expose the ProvideServerData function, or if the parameters of the function do not match, no server data will be provided and the error will be ignored.

To retrieve Server Data on the client, you must parse the .DLSC file. Server Data can be found by searching for the ServerData tag in the .DLSC file. The first child contains the Base64 encoded string representing the binary value.

The sample code shown also shows how to convert the binary data back into a string. If you used a serialized object, you would (of course) call the appropriate deserializer here.

```

[VB]
Dim xdoc As New XmlDocument()
Dim xlist As XmlNodeList
Dim data As String

xdoc.Load("file path here")
xlist = xdoc.GetElementsByTagName("ServerData")
Try
    data = xlist.Item(0).ChildNodes(0).Value

    Dim result As String
    Dim bytes() As Byte
    Dim converter As New System.Text.UnicodeEncoding()

```

```

        bytes = Convert.FromBase64String(data)
        result = converter.GetString(bytes)
    End Try
[C#]
    XmlDocument xdoc = new XmlDocument();
    XmlNodeList xlist;
    string data;
    xdoc.Load("file path here");
    xlist = xdoc.GetElementsByTagName("ServerData");
    try
    {
        data = xlist.Item(0).ChildNodes.Item(0).Value;
        string result;
        Byte[] bytes;
        System.Text.UnicodeEncoding converter =
            new System.Text.UnicodeEncoding();
        bytes = Convert.FromBase64String(data);
        result = converter.GetString(bytes);
        textBox1.Text = result;
    }
    catch
    {}

```

The information shown here illustrates how you can provide additional information to use during client licensing. You could use it to specify a limit to the number of clients, or to override the demo expiration date, or set an expiration date for the DLSC file, or to otherwise provide any other kind of licensing functionality that depends on data from the server.

The Licensing System includes support for encrypting and decrypting `ServerData`. In the previous sample project, you would add a reference to the `Desaware.MachineLicense.dll` assembly file, then change the `ProvideServerData` function to the following if you want to return encrypted `ServerData`.

```

[VB]
Public Function ProvideServerData(ByVal appname As String, _
    ByVal UniqueGuid As String, ByVal InstallationCode As String, _
    ByVal CustomData As Specialized.StringDictionary) As Byte()
    Dim result As New Text.StringBuilder()

    ' build your ServerData string as before using the result variable
    :
    :

    Dim ClientLicense1 As New ClientLicense()
    Dim subiv() As Byte = {11, 36, 212, 124, 48, 10, 216, 79, 95, 19, 241,
184, 26, 43, 150, 151}

    Return ClientLicense1.EncryptServerData(appname, subiv, result.ToString)
End Function

[C#]
public byte[] ProvideServerData(string appname, string UniqueGuid, string
InstallationCode, System.Specialized.StringDictionary CustomData)
{
    System.Text.StringBuilder result = new System.Text.StringBuilder();

```

```

        // build your ServerData string as before using the result variable
        :
        :

        ClientLicense ClientLicense1 = new ClientLicense();
        byte [] subiv = {11, 36, 212, 124, 48, 10, 216, 79, 95, 19, 241, 184, 26,
43, 150, 151};

        return ClientLicense1.EncryptServerData(appname, subiv,
result.ToString());
    }

```

Similarly, when decrypting encrypted ServerData, use the ClientLicense class's GetEncryptedServerData function.

[VB]

```

Dim xdoc As New XmlDocument()
Dim xlist As XmlNodeList
Dim data As String

xdoc.Load("file path here")
xlist = xdoc.GetElementsByTagName("ServerData")
Try
    data = xlist.Item(0).ChildNodes(0).Value

    Dim result As String
    Dim subiv() As Byte = {11, 36, 212, 124, 48, 10, 216, 79, 95, 19,
241, 184, 26, 43, 150, 151}
    result =
ClientLicense1.GetEncryptedServerData(ClientLicense1.ApplicationName, subiv,
data)
End Try

```

[C#]

```

XmlDocument xdoc = new XmlDocument();
XmlNodeList xlist;
string data;
xdoc.Load("file path here");
xlist = xdoc.GetElementsByTagName("ServerData");
try
{
    data = xlist.Item(0).ChildNodes.Item(0).Value;
    string result;
    byte [] subiv = {11, 36, 212, 124, 48, 10, 216, 79, 95, 19, 241,
184, 26, 43, 150, 151};
    result =
ClientLicense1.GetEncryptedServerData(ClientLicense1.ApplicationName, subiv,
data);
}
catch
{}

```

Refer to the FriendlySecurityServerData and CustomServerData sample projects for more details.

Additional ServerData Considerations

It is extremely important to understand the nature of the UniqueGUID parameter provided to the ProvideServerData function. The UniqueGUID value is assigned to each unique system that is identified by the licensing system. But what is a unique system?

The licensing system assigns a unique GUID to a certificate based on the installation match algorithm. As described earlier under system matching algorithms, given the fact that hardware can change, and algorithms can vary (if you create custom algorithms), it is very possible for two systems to end up with the same UniqueGUID value. It is also possible that the Custom Data in the database will not reflect the current installation (in cases where the Installation Match algorithm passes, but the Demo Match algorithm fails – custom data is only updated in the database when the higher standard of system match is present).

In order to provide you additional flexibility in identifying systems, the ProvideServerData function includes a CustomData parameter. This parameter is a StringDictionary containing the Custom Data that was actually sent in the current installation request.

You can have your client code add additional custom data that identifies the installation, or user, and use that information to help generate the Server Data. Note, however, that changes to the contents of this parameter will not modify the custom data in the certificate.

The rule is this: Custom Data is always set on the client and is not modified by the server. Server Data is always set on the licensing server, and is not modified by the client.

The Desaware Licensing System: Interview with the Architect

Rather than the traditional, somewhat “dry” approach for introducing a product, we thought we’d try something different. In this article you’ll find an interview with Dan Appleman, architect of the Desaware Licensing System, in which he discusses the philosophy and design of the system, along with common licensing scenarios and tradeoffs involved.

What prompted you to develop a new licensing system?

Like virtually every product we develop, it starts out with a problem that we or our current customers need to solve. We had received a number of requests for a licensing scheme that would license software to a particular machine or server. And with the development of our own CAS/Tester and 5-Minute Software line, we also needed this capability. Rather than come up with a custom solution, we decided to come up with a system that would be adaptable to a variety of scenarios and offer it to our customers.

So you came up with a flexible system that could be adapted to any Windows licensing need?

Oh no! Definitely not. When it comes to licensing, the more features you have, the more chances you’ll have some kind of security hole. No, our focus was on security, and ease of use and deployment. There is flexibility, and definitely extensibility, but the base system just supports a particular set of scenarios very well.

I want a really secure licensing system that will make it very difficult for someone to pirate my code. Will your system do this?

Yes. You can be extraordinarily strict and secure.

I want a very friendly licensing system, that will tolerate system changes, work within an Internet connection, and be as “friendly” as a licensing system can be. Will your system do this?

Yes. But remember – the more tolerant you are, the easier it is to bypass your licensing. But that’s really a matter of your corporate philosophy.

We include implementations for several different scenarios with the product, ranging from very strict, to very tolerant.

And, for those who are interested, Desaware uses the more tolerant licensing philosophy.

One thing I noticed is that you only support licensing of .NET components and applications. Why is that?

It’s not just because we are focused on .NET now. But to explain it I’ll have to talk a bit about more traditional licensing approaches.

Here’s the first thing you have to realize. Any licensing system that is not based on server activation is breakable.

Any licensing system not based on server activation is breakable?

It's true. Sure, you make it hard to break the security, but no matter how good the encryption on the installation code you use, there's nothing to prevent someone from using it on multiple systems unless you have an outside secured system to keep track each time an installation code is used and which system it's used on.

Traditional licensing schemes use a variety of techniques – from hidden files to obfuscated registry entries to proprietary services (which are rather too invasive on a system for my taste). They are all based on secrets.

Secrets can be discovered. Security can be bypassed by modifying (cracking) the licensed application or the licensing software itself.

I wanted a system that is end to end cryptographically secure – meaning that the level of security depends on the strength of the cryptography and not on any secret information.

How are .NET assemblies different from other Windows components and applications?

The .NET framework allows developers, for the first time, to cryptographically sign each assembly. This happens any time you give an assembly a “strong name” using a private key that you create. There are two important things about strong names: First, it is impossible to modify a strong named assembly. If you try, the .NET runtime will detect the change and refuse to run the assembly. Second, it is impossible to tamper with a dependent assembly. In this case it means that once you use our licensing component to add licensing to your assembly, it is virtually impossible to bypass it. You can't modify the licensed assembly. You can't modify our licensing component. And you can't swap out our licensing component for another.

You used the term “Cryptographically Secure”. I also noticed you said “Virtually impossible”. Exactly how secure are we talking about?

Cryptographically secure means two things. First, that the level of difficulty depends on the strength of the cryptographic algorithm and key. We use 128 bit encryption in the Desaware Licensing System, which means there are 2^{128} possible keys. Finding the correct key would take longer than the life of the Universe using today's technology, so the real risk in breaking this security is that someone will steal your key.

It also means that the security depends purely on the strength of the key, and not on any secret information. You should be able to publish the source code for the licensing system and not compromise the security. In fact, we do offer a source license to the Desaware Licensing System for those who want it.

You mentioned that secure licensing requires connection to a server, does that mean you have to have an Internet connection to activate software licensed with our system?

Not necessarily. If you want cryptographically strong security, then yes – you have to have a connection to a server. But we also wanted to support other scenarios – cases where an Internet connection isn't allowed or available, or where you want to support cases where the licensing server is down. So, for those who are willing to accept weaker security, we support deferred

activation. When you install licensed software our licensing component creates a temporary license. It will try to obtain a permanent license each time you run the program. It's up to you to decide what to do if it can't obtain a license. You could do a timed activation – like Microsoft, where you have to activate within a certain set time like 30 days. Or you can simply allow the software to keep running on the temporary license forever.

We also have a mechanism that allows you to validate temporary licenses received via other channels such as Email or even floppy disk, allowing you to enforce cryptographically secure licensing even if an Internet connection is unavailable.

Changing the Subject, you mentioned that a licensing system has to track use of installation codes and which systems they are installed on. Doesn't this compromise your end user's privacy?

That was actually one of the critical design issues. I wanted a system that could determine if an installation code was being used on multiple systems, but which would not compromise the privacy of the computer on which the licensed software is installed. Fortunately, cryptography provided an answer to this problem as well. Rather than basing system identification on information from the client system, we based it on cryptographic hashes of information from the client system.

To give you an idea of how this might work, let's say one of the ways a system is identified is by the name of the computer, and the computer is named "myComputer". We don't actually send the word "myComputer" to the licensing server, we send a 256 bit hash of that value. This allows us to identify the computer in the license server database, but it would be impossible to go back and obtain the computer name given the hash value.

What kinds of system information do you use to identify a system?

By default we use the computer name, system disk volume ID, and the addresses of any Internet adapters. However, you can extend the system by writing your own code to identify systems. You can add other identification information, and define your own algorithm to determine whether two systems are identical.

What about demonstration or trial installations?

The Desaware licensing system does support timed demonstration installations, where you can define the duration of the demonstration period. Like installations, the demo installation connects to the licensing server to provide high security. For example: if a demo installation expires and you uninstall it, any attempt to reinstall the demo on the same system will be flagged by the server and you'll see a demo-expired error. Connecting the server also ensures that the demo certificate is based on the server date, reducing the chance that someone can create a longer demo period by changing their system date.

Can you give me an idea of how the Desaware Licensing System works? What is the typical sequence of events?

The system consists of two parts: The client licensing component (Desaware.MachineLicense.Dll) and the server licensing service (Desaware.ServerLicense). The

service is a .NET web service that can run in a virtual directory of any IIS system that has the .NET framework installed.

There is also a license manager application that works with the web service to allow you to manage the licensing system (create applications, installation codes, etc.).

Your first step in licensing an application is to use the license manager to define the application.

The license manager also creates a resource file that your developers will use. This resource file contains the application's public key and some other information. You'll also use the license manager to create installation codes.

Your developers will then add the client component to the applications to be licensed. This is a very simple process – you just create the component, and add a few lines of code to verify if a license exists.

The developers also decide how the client will install the license – where will the installation key be entered by the user. You can do this during installation, or allow users to enter installation keys into existing demo installations.

When the user installs the licensed product, they enter the installation code that came with the product (one you created using the license manager). The licensing component then connects to your web service and makes sure the installation code is valid, and that the code has not been previously used on another system. If licensing is successful, a digitally signed certificate is installed on the local system. When the licensed application runs it checks the certificate, and makes sure it matches the current system and has not been tampered with.

There are variations available, for example: whether or not you support demonstration mode, how you deal with license violations and what you consider a license violation, and what you do when there is not Internet connection, but these are easy to define and handle.

So each installation code can only be used on a single system?

Oh no – I hope I didn't give that impression. Sure, you can set it up that way, however we prefer to give people a bit more flexibility. You can set two numbers – a warning number and a blocking number. The warning number is the number of computers that can use the same installation code before the server returns a warning. With a warning, the server still returns a valid license, but it returns a result letting you know that a violation may be occurring. The blocking number is the number of computers that can use the same installation code before the server rejects the installation code.

What is the maximum number of computers that can share an installation code?

There's no practical limit. In fact, for site licenses you might prefer to give a customer a single installation code to use on all their systems rather than giving them a large number of installation codes. And yes, you can easily add to the number of computers that can use an installation code.

Let's say a user installs software on a system, then copies the directory containing the application to another system. Will the application work?

Probably not. If the other computer has a matching system identifier, and your computer matching algorithm allows it, it will run, but our default algorithm is such that it is extremely unlikely that two different computers will match. And the license certificate is bound to a specific system.

What if someone tries to modify the license certificate?

The license certificate returned from the server is digitally signed using the private key of the application. So any attempt to tamper with the certificate will result in an “invalid certificate” error which is considered a license violation.

What happens if a license violation occurs?

That's up to you. The client license component does not include a user interface, or decide for you how you want to respond to a license violation. We do include samples of several scenarios.

What if I want two versions of a component or several executables to be licensed together - using the same installation code?

No problem at all. The Desaware License Server works with what we call “applications”, which is a name you define. Every assembly that uses the same application name can share installation codes and license certificates. If you revise a component, and want it to continue to use the previous installation code and license certificate, just use the same application name.

The .NET Framework has a licensing scheme for components based on the LicenseProvider class. Does your system use this?

The Desaware Licensing System does support license providers for components if you wish. That's a good approach for components which are licensed only for design time and are redistributable at runtime. We support license providers as part of our licensing system, however the core system is designed for per machine/server licensing and does not use the require use of .Net license providers.

My customers are developers who often have multiple operating systems on their system (multi-boot). Do they need a different Installation Code for each partition?

Probably not. The default system identification algorithm we use includes the network card address which is the same under each OS. So your customers can just reinstall your software under each partition and their same installation code will work. You can change this by defining your own system identifier and matching algorithm.

Can I do per-user licensing?

We do not support this feature by default. However, you can define a system identifier that is based on the user or account information, and then define a system matching algorithm that

considers each user to effectively be a different licensed machine. This is demonstrated in the example that illustrates how you can extend the licensing system.

Can I do per-feature licensing?

This feature allows you to have a single application support several features, each of which is enabled independently. The Desaware licensing system supports this easily. Just as you can have multiple assemblies licensed as a single application, you can have one assembly licensed as multiple applications. In other words: you define a new “application” in the licensing system for each feature you want to enable. In your assembly, you will have a resource file and instance of the licensing component for each feature – and each feature will have its own license certificate. You can then verify each one independently.

How is the Desaware Licensing System itself licensed? How many applications can I license? Are there any hidden costs?

I actually saw a licensing product once that wanted to charge a percentage of revenue of the licensed software. Hard to believe, isn't it?

Our story is much more simple and quite economical. The Desaware Licensing System is itself licensed using its own technology.

The client license component and License Manager application is licensed on each developer or administrator seat. There is no redistribution fee for including the component with your licensed applications.

The licensing service is licensed per server. Each service can support an unlimited number of applications.

In practice, you need a minimum of one server and one client license.

We also offer site licenses and source code licenses.

Do I need a secure server (SSL) for my license server?

No. All information between the client component and the server is encrypted by the components themselves. There is no need for connecting to the server via SSL. It will only reduce performance.

How long is your installation code?

26 characters. Yes, I know it's long, but we wanted to support full 128 bit encryption across the board. The good news is that the code is limited to upper case characters, and excludes certain potentially confusing letters like I and O (which can be confused with 1 and 0). We logically divide into 5 groups of 5 characters, plus one trailing character. And we include an installation code control which you can drop into your user interface to acquire the code.

Can I collect additional information (such as registration information) during the licensing process?

Yes you can. We call this “custom data”, and you can add this to the licensing certificate and it will be sent to the server during the activation process. Remember that any information you collect should adhere to your organization's privacy policy. Our license manager allows you to

view this custom data. We also publish our database schema so you can access the data directly. And there's a mechanism for the server to execute a component you provide every time an installation occurs.

Can I license windows form controls or other software intended to be downloaded via Internet or intranet?

The .NET framework runs downloaded software with limited permissions, also known as "partial trust". The Desaware Licensing System can be used to license partially trusted code, but only if the licensing component is first installed in the global assembly cache as a fully trusted component.

What version of the .NET Framework do you require?

We include components build with both the 1.1 and 2.0 versions of the Framework. Either version of the client licensing component can communicate with either version of the licensing server. You must use the latest 1.1 server components if you wish to support the 2.0 framework client component..

Are there any other requirements for the Desaware Licensing System?

Yes, but there aren't many.

- Obviously, both client and server software must be running the .NET framework.
- The server must be on Windows 2000 or greater running Internet Information Server with ASP.Net.
- You need a database on the server. The licensing server works with SQL server, MSDE, or JET (Access). You can also use any OleDb compliant database (or ODBC if using the 1.1 or later version of the .NET framework)
- 128 bit encryption. Sorry, we don't support a 40 bit option.

Tell me some of the things the Desaware Licensing System can't do.

No software package can do everything, and we'd rather you know the limitations up front.

- This form of licensing is not good for components that are intended to be embedded into other components. That's both because the .NET component licensing system is not particularly good at handling this case, and because it wasn't a design requirement for the current version of the product. So if you're creating a control intended to freely distributable as part of another control that has a different licensing scheme, you should look elsewhere. However, if you're creating a control or component for use on Windows forms, web forms, etc. and you want per machine/ per server licensing (either always, or just at design time), this system will work just fine.
- It won't work in France. Wait, that's not entirely true. France has rather severe limitations on use of cryptography, thus portions of the Windows encryption system simply don't work on French versions of Windows. So the licensing server must run on a version of Windows that supports public/private key encryption. The system is designed

to work with clients that run on French Windows, with the limitation that communication with the licensing server is not secure.

- We don't implement concurrent Licensing (where you allow a certain number of systems to run simultaneously). This feature is not supported in the current version of the Desaware licensing System. One reason is performance and reliability – we wanted a system that only needed to connect to the licensing server once, not each time the program ran. You can create your own web service that tracks concurrent users that is based on our licensing system (using information from the license certificate, and tying into the licensing database on the server), but we do not yet provide an implementation that does this.

More FAQ's

Why don't you have a trial installation that counts the number of times an application is used?

You can implement such a scheme if you wish, but unless you connect to the server each time the application is run, it isn't possible to make such an approach cryptographically secure. Any scheme that hides the usage count on the local system is breakable. So we did not build in support for this approach.

Is an Installation Code Reclaimed when an application is uninstalled?

No. Think about it – if you allowed an installation code to be reclaimed when software was uninstalled you would have no security at all. Anyone could do an install, save the directory or partition, do an uninstall, then restore the directory or partition and bypass your security completely.

The License Manager and Licensing web service make it easy for you to reenable an installation code if you wish. But you'll need to add dealing with this situation into part of your customer service program.

What happens if I reinstall software on a licensed system using a different installation code?

If the demo system matching algorithm determines that the system is identical to one seen before, the new installation code will replace the existing one in the licensing database. However, if there is any doubt, the system will end up using two installation codes. So you should try to avoid using different installation codes on the same machine.

What are the most important things I should do to secure my license service?

1. Make sure the Management.asmx web service entry point is secured either by IP filter or by roles. The default installation enables access only by the server itself.
2. Limit access to the licensing database. That's where the application passwords and private keys are kept – you don't want anyone outside to get a hold of it.

We do a number of things to help protect you even if outsiders can connect to the management service. For one thing, there is no way to retrieve the private key from the database using the service. Also, the service does not delete information from the database (except for replacing custom data under certain circumstances).

What if someone installs a demo/trial version on a system that already has (or had) a full installation?

In most cases you'll get a demo expired error. When you do a full installation on a clean system, it writes the current date into the database as the expiration date, thus prohibiting later demo installs. However, if you start with a demo install, then do a full install, then do a demo install, it

should succeed with a warning, and the expiration date will be based on the first time the demo was installed (this, of course, applies to a scenario where the server is accessible).

Oh no! Someone hacked my system and copied my licensing database! Is all lost?

Well, if they deleted it you're in trouble. People with valid install codes won't be able to use their software. So you should be sure to keep backups (which you do because you backup your server regularly, right?)

But even if they have a copy of your database, things aren't too bad. Yes, they can generate keys that will pass the first pass verification, but they still can't add new keys to your database or work with it as long as the Management.asmx file is secured. They can't redirect software you ship to a different server because the licensing URL is typically hard coded into your software. They would be able to generate fake installations for installation codes they know about, but that's probably more an annoyance than serious problem.

All you need to do in this case is ship out a new version of your software using a newly created application. Because each application has its own private key, the damage to losing one is limited to that application.

Appendices

Glossary

Desaware Licensing System – The name of this product.

Licensing Server – The web service used implement the licensing system.

License Manager – The Windows forms application used to manage applications and licenses. It connects to the Licensing Server.

The Machine License Component – Desaware.MachineLicense.Dll. The component used in your licensed applications and components. Use it to both install and verify licenses.

Application – Any group of one or more assemblies that are licensed together. The name of the application bears no relation to the assembly names. The same license certificate will enable all assemblies that belong to an application. More than one version of an assembly can be part of an application. Different versions of an assembly can belong to different applications.

License Certificate – (DLSC file) This is an XML document that is used by the licensing system to indicate that a specific application is licensed for a specific machine. It can be temporary or signed.

Signed License Certificate – This is a License Certificate that has been digitally signed by the server to indicate that it is valid.

Temporary License Certificate – This is a temporary certificate installed when the server cannot be reached. Use of temporary certificates is optional.

Demo License Certificate – A license certificate that does not contain an installation code.

Configuring the Web Service web.config File

The license server web.config file contains a number of entries in the appSettings section that are used by the license server. Each one follows the standard application configuration file schema of an <add key=... value=> tag.

The following keys are supported:

ipfilter	A list of IP addresses that can access the Management.asmx entry point. The format is a.b.c.d[-e.f.g.h], where each entry is a single or range of IP addresses. Entries may be separated by semicolons. If no ipfilter entry is found, the local system (127.0.0.1) is assumed.
allowedroles	One or more roles in the form system domain\user role indicating the accounts allowed to access the Management.asmx entry point. If none are specified, only the ipfilter is used.
connectiontype	The string “oledb”, “sql” or “odbc” (>1.1. Framework)

	indicating the type of database provider to use.
connectionstring	The connection string used by the license server to connect to the licensing database.

Read the section “Testing and Debugging the License Server Installation” for additional information on modifying default settings of the web.config file.

Database Schema and Contents

You may wish to connect directly to the database for a variety of reasons, including extracting registration data, or integrating the licensing system into your own database.

Please note that the default configuration of the database does not enforce referential integrity – so we strongly suggest you do not modify any entries in the database unless you really know what you’re doing.

This document limits itself to a description of each column. You can examine the database directly for information on field types and sizes. Each table includes an ID column as primary key. This column is not used by the licensing server.

Application Table

ApplicationName	Name of the application
ApplicationDescription	Description
WarningCount	Number of installs allowed before a reuse warning.
BlockCount	Number of installs allowed before a code is rejected.
DemoExpirationDays	Default and max number of days a demo version is valid.
ApplicationPassword	Password used for first pass verification and temporary certificates.
SignatureKey	Public/Private key
Identifiers	Reserved for future use.

InstallationCodes Table

CodeGUID	GUID value of this installation code.
ApplicationName	Name of the application for this code.
WarningCount	Warning count for this code. Default value is set from the application table.
BlockCount	Block count for this code. Default value is set from the Application table.
UserDefined	Available for your use. Up to 255 characters.
CodeStart	First 5 letters of the installation code in normal format.

Used for searches.

UniqueInstalls Table

UniqueInstallGUID	GUID identifying a specific installation.
DemoExpiration	Expiration date if demo.
InstallationDate	Date of first install.
CodeGUID	Installation code used on this installation. Null for demo install.
ApplicationName	Name of the application installed.

SystemIdentifiers Table

Name	Name of this system identifier.
Value	Value of this system identifier.
UniqueInstallGUID	GUID identifying a specific installation.

CustomData Table

CustomName	Name of this custom name.
CustomValue	Value of this custom name.
UniqueInstallGUID	GUID identifying a specific installation.

Installation and Existing Certificates

What happens when you do an installation on a system where a license certificate already exists?

There are eight possible certificates that can be installed.

- Valid demo cert (temp)
- Invalid demo cert (temp)
- Valid demo cert (signed)
- Invalid demo cert (signed)
- Valid full cert (temp)
- Invalid full cert (temp)
- Valid full cert (signed)
- Invalid full cert (signed)

There are four installation cases:

- Demo install activation required.
- Demo install activation optional.
- Full install activation required.
- Full install activation optional.

Let's consider each option in detail.

For activation optional results, there are fatal and non-fatal errors. A fatal error is one in which the server rejects the license. A non-fatal one is one in which the server cannot be reached.

Fatal errors include:

- Invalid certificates sent from the client.
- Demo has expired.
- Code reuse is blocked.
- Install code is invalid.

Demo install, activation required:

Existing Certificate	Success	Failure
Valid demo cert (temp)	Overwrites existing	Leaves existing
Invalid demo cert (temp)	Overwrites existing	Leaves existing
Valid demo cert (signed)	Overwrites existing	Leaves existing
Invalid demo cert (signed)	Overwrites existing	Leaves existing
Valid full cert (temp)	Leaves existing	Leaves existing
Invalid full cert (temp)	Leaves existing	Leaves existing
Valid full cert (signed)	Leaves existing	Leaves existing
Invalid full cert (signed)	Leaves existing	Leaves existing

In other words, if you do a demo install on a system that the server believes is already fully registered, the software will assume that any existing certificate is that full license certificate, and will leave it in place. The software does not actually check what kind of certificate is currently on the system – it bases its information purely on the server records, so if the certificate is not a full certificate or is invalid, doing a demo install will not overwrite it. You can override this behavior by verifying the existing certificate and deleting it before doing the install.

Demo install, activation optional:

Existing Certificate	Success	Deferred	Fatal Error
Valid demo cert (temp)	Overwrites existing	Installs temp cert	No Cert
Invalid demo cert (temp)	Overwrites existing	Installs temp cert	No Cert
Valid demo cert (signed)	Overwrites existing	Installs temp cert	No Cert
Invalid demo cert (signed)	Overwrites existing	Installs temp cert	No Cert
Valid full cert (temp)	Overwrites existing	Installs temp cert	No Cert

Invalid full cert (temp)	Overwrites existing	Installs temp cert	No Cert
Valid full cert (signed)	Overwrites existing	Installs temp cert	No Cert
Invalid full cert (signed)	Overwrites existing	Installs temp cert	No Cert

An activation optional demo install always overwrites the existing certificate with a temporary certificate before contacting the server. If a non-fatal error occurs, that temp certificate remains installed. Otherwise the temp certificate is deleted and the software is unlicensed.

You can override this behavior in your install program by checking first for a valid license and not allowing the demo install if the full license is present.

Full install, activation required:

Existing Certificate	Success	Failure
Valid demo cert (temp)	Overwrites existing	Leaves existing
Invalid demo cert (temp)	Overwrites existing	Leaves existing
Valid demo cert (signed)	Overwrites existing	Leaves existing
Invalid demo cert (signed)	Overwrites existing	Leaves existing
Valid full cert (temp)	Overwrites existing	Leaves existing
Invalid full cert (temp)	Overwrites existing	Leaves existing
Valid full cert (signed)	Overwrites existing	Leaves existing
Invalid full cert (signed)	Overwrites existing	Leaves existing

Full install, activation optional:

Existing Certificate	Success	Deferred	Fatal Error
Valid demo cert (temp)	Overwrites existing	Installs temp cert	No Cert
Invalid demo cert (temp)	Overwrites existing	Installs temp cert	No Cert
Valid demo cert (signed)	Overwrites existing	Installs temp cert	No Cert
Invalid demo cert (signed)	Overwrites existing	Installs temp cert	No Cert
Valid full cert (temp)	Overwrites existing	Installs temp cert	No Cert
Invalid full cert (temp)	Overwrites existing	Installs temp cert	No Cert
Valid full cert (signed)	Overwrites existing	Installs temp cert	No Cert
Invalid full cert (signed)	Overwrites existing	Installs temp cert	No Cert

Working with proxy servers

The Desaware.MachineLicense component uses SOAP over http to communicate with the server. It is essential that any firewalls be configured to allow SOAP headers to pass.

By default the licensing component uses the default proxy settings for Internet Explorer. However, you can manually configure the proxy server settings for your application in its application configuration file (assuming you provide one). This file can be edited with any text editor or XML editor.

It is essential that the modifications be done exactly as shown.

Find the end of the file. Before the last line (</configuration>) add the following:

```
<system.net>
  <defaultProxy>
    <proxy autoDetect="False" bypassonlocal="True"
proxyaddress="http://proxyaddress:proxyport" />
  </defaultProxy>
</system.net>
```

Edit the autoDetect, bypassonlocal, proxyaddress and proxyport elements as follows:

autodetect – Specifies whether the proxy is automatically detected

bypassonlocal – Specifies whether the proxy is bypassed for local resources (such as 127.0.0.1, <http://localhost>, etc.)

proxyaddress:proxyport – Specifies the proxy URI and port to use.

When you restart the application, the manual proxy settings will take effect for this application.

If your proxy server authenticates connections, you may need to modify the

<defaultProxy> element to be <defaultProxy useDefaultCredentials="True">

Version History - 1.2 Update

The version 1.2 update is identical to version 1.1 except that it includes native components for .NET 2.0. The .NET 1.1 client components are fully compatible with the .NET 2.0 server.

To use the .NET 2.0 client components with the 1.1 server you MUST install the updated 1.1 framework server components. This is necessary to resolve a change in the .NET framework in the way dates are serialized. The .NET 2.0 client components are not compatible with the 1.0 server. This also applies to older MachineLicense components running under the .NET 2.0 framework. The .NET 1.1 MachineLicense component included with this release is a drop-in replacement for the previous version and will work under the .NET 2.0 framework with the updated server.

The remainder of this section refers to the 1.1 update.

License Server

- Improved web server diagnostics helps resolve unusual configuration and installation issues. Supports toggling of diagnostics in the web.config file. **Note that the web.config file now has diagnostics turned on by default. You need to disable this when done testing (comment out the key <add key="enablediagnostics" value="true" />).**
- New GetServerDate function added to the web service Activator.asmx to return a DateTime object based on the server date.

MachineLicense

- New InstallationDate property in the ClientLicense object. This property is valid after a call to the VerifyLicense function. The InstallationDate property returns the installation date recorded in the license certificate file.
- New ExpirationDate property in the ClientLicense object. This property is valid after a call to the VerifyLicense function. The ExpirationDate property returns the demo expiration date recorded in certificate file.
- New DemoRequireInternet property in the ClientLicense object. This property allows you to require an internet connection for demo licenses. If this property is set and a demo license exists, the ClientLicense object will attempt to retrieve the date and time from multiple servers on the internet when the VerifyLicense function is called. It will use the internet time rather than the system time to compare whether the demo has expired or not.
- New DemoNoInternetDate ValidationStatus result. Returned when a demo certificate is verified, the DemoRequireInternet property is set and the internet time servers cannot be reached.

Samples

- Updated High Security and Friendly Security samples to demonstrate use of the new features.
- New sample demonstrating how to implement "module licensing" based on multiple "Applications".
- Advanced sample of System ID and System match algorithm to make the software more secure.
- Example showing how to retrieve CustomData (XML data) from the certificate file (registration information).
- New sample demonstrating how to encrypt and decrypt Server Data.

Utilities

- New License Manager includes many new features including:
 - Support for integrated Windows authentication
 - Ability to authenticate remotely thru a web service.
 - New BindingCheck code generator utility – additional code verification that your assembly has not been tampered with.
- Application Note on how to prevent reverse engineering and additional strong name checking to verify that your assembly has not been modified.
- Includes new SystemID utility – displays SystemID information as hash values. Will allow you to match SystemID information on your customers systems without requiring them to disclose private system information.
- Includes the compiled edition of the QND Obfuscator – protect your code from decompilation.

Single Application Edition

Version 1.1 introduces a new lower cost edition of the licensing system. This version has the following limitations:

- Supports a single application (the standard licensing system supports licensing of unlimited applications).
- Allows up to 1000 installation codes (the standard licensing system has no limit).
- Does not support signing of external DLSC files.
- Does not support server data or post install actions (licensing server extensibility).

The single application edition is easily upgraded to the full edition. See the section on Advanced Menu/Hosted Install for details on upgrading.