The Desaware Licensing System

Application Note #2:
Implementing Concurrent/Floating Licenses

Version 1 – February 2009

Introduction

The Desaware Licensing System (DLS) emphasizes flexibility and extensibility. It is designed to be an infrastructure product that can be easily integrated into existing business systems.

One of the most frequent requests we receive is for concurrent or "floating" licensing – the ability for software to be installed on many systems but only run on a limited number of those systems simultaneously.

This application note contains a sample implementation of a floating license system built upon the Desaware Licensing System infrastructure. We recommend you first read Application Note #1: Extending the Licensing System. This implementation is built upon and compatible with the second edition of that application note[1].

The examples in this application note are provided in VB .NET. The examples are simple, and we expect any .NET developer will be able to easily read them regardless of their own preferred language.

Sample Installation

Unzip the application note file into the directory of your choice. The samples consist of the following directories:

- extensions – contains the project for the Desaware.extensions.dll project.
- server – contains a licensing server web application for testing.
- expirations – contains a web application and web service that demonstrate the examples in this application note.
- HighSecurity – contains a modified version of the HighSecurity sample application that demonstrates how to implement concurrent licensing.
- ConcurrencyServer – contains the concurrency "floating license" server.
- db – directory holding the licensing database and log files.

To configure the samples, do the following:

- Reset the security on the db database to give full access to the ASPNET or NETWORK SERVICE accounts. Do the same for any files contained within this directory.
- In the server\bin directory, replace the DLS10Server.dlsc file with the one from your own licensing server on the same system. If you have not installed the server on this system, download the licensing server demo from Desaware.com, install it as a demo, then copy the DLS10Server.dlsc file created during that installation into this directory. The certificate will remain valid for one month.
- In the HighSecurity\bin directory, replace the DLS10Client.dlsc file with the one from your own licensing system client on the same system. If you have not installed the client on this system, download the licensing client demo from Desaware.com, install it as a demo, then copy the

[1] Please download the latest version of Application Note #1: Extending the Licensing System. It was updated to version 2 to make it compatible with this application note.

DLS10Client.dlsc file created during that installation into this directory. The certificate will remain valid for one month. On Vista this file might be found in the hidden ProgramData directory.

- In the server\web.config file, set the following paths to the correct location on your system:
  - In the <appSettings> section, look in the "connectionstring" key.
  - In the <system.diagnostics> section, look under <listeners> and set the initializeData property for the DlsWriterListener term.
- Make the same web.config changes to the expirations\web.config file.
- In your IIS administration panel, add a virtual directory to the directory in which you installed the samples. Open the properties for the directory and make sure the ASP .NET tab is set to ASP .NET 2.0.
- Using the IIS administration panel, navigate to the virtual directory you just created. Right click on the expirations directory to view its properties, and select the"Create" button to turn the directory into a separate web application. Repeat this step for the server directory, and for the ConcurrencyServer directory.
- Assuming you name the virtual directory dlsappnote2, you should be able to access the license server at http://localhost/dlsappnote2/server/management.asmx, the expiration utility at http://localhost/dlsappnote2/expirations/dlsextdefault.aspx and the concurrency server at http://localhost/dlsappnote2/concurrencyserver/licenser.asmx.
- Using your LicenseManager application, connect to the new server at http://localhost/dlsappnote2/server/management.asmx. Select "Developer Information" in the left panel and SaveInfoAsResX file on the right panel. Save the new resx file in the HighSecurity directory. This will cause the application to refer to the correct server on your system when testing this example.

You should now be able to run and build all of the examples using Visual Studio 2008.

Concurrent Licensing Server Design

Concurrent licensing is tricky. Doing it with any degree of security is even trickier. This section will review the requirements and architecture of any concurrent licensing solution, and the specific implementation of the DLS solution.

A Central Concurrency Server

In a floating license scheme, software is installed on multiple machines. Before running, the software has to contain a mechanism to determine how many copies of the software are currently running. That means there must be a way for software instances to communicate with each other.

Peer to peer solutions are theoretically possible, but extraordinarily complex, as you have to somehow guarantee that every possible machine can reach every other machine all of the time. Any communication failure could cause extra licenses to be granted.

A more reliable, secure and simple solution is to define a central license server that grants individual machines permission to run and keeps track of how many clients are running at any given time.

As part of the security implications of this approach it is necessary to consider the following:

- The server needs to ensure that client requests are valid.
- Clients must connect to the correct server.
- The server must be designed for high performance.

A Leasing System

In a floating licensing scenario, an application will typically contact the concurrency server and request a “lease”. A lease is essentially permission to run. The concurrency server holds a certain number of leases based on the number of licenses (applications allowed to run simultaneously). When an application closes, it is expected to return the lease to the server so it will be available for other applications.

As part of the leasing system, it is necessary to consider the following:

- It is possible for applications to crash or hang, in which case they might not return the lease to the server. Thus leases must have an expiration time after which they are presumed returned.
- Because leases can expire, there needs to be a renewal mechanism so applications can let the server know not to expire the lease.
- Because applications will be contacting the server to renew leases, the server must be optimized for performance to handle the expected load.

Server Hosting and Communication

One key question is where the server is hosted.

One option would be to have a single centralized server on the same host as the licensing system server. This would be a simple solution in that the central server could have direct access to the licensing system database for verification purposes. However, this approach would preclude deployment of the concurrency server to customer sites, and would impact scalability.

For this reason in our implementation the concurrency server is a completely independent piece of software. Moreover, a separate concurrency server can be set for each installation code or group of installation codes (necessary in cases where a server might be deployed at a customer site). Scalability is easily achieved in that you can deploy as many servers as you need.

Because the concurrency server is an independent piece of software, all information it needs to operate must come from the clients. This can be done securely under the DLS because each client has a license certificate that has been digitally signed by the licensing server. This certificate contains all of the information the concurrency server needs in order to handle a particular installation code.

The server checks the signature date of each certificate, updating its internal records based on the most recent certificates.

The Concurrency Server Methods and Operation

The concurrency server is accessed via web service calls.

AllocateLicense

Parameters:

- cert – string
- LicenseLifetime – Integer
- InstallCode – string (reference)
- UniqueCode – string (reference)
- Returns ConcurrencyResults enumeration

The AllocateLicense function is used by the client to request a lease. The cert parameter contains the full XML certificate for the licensed client. The LicenseLifetime parameter contains the requested lease lifetime (in seconds). The application must renew the lease before this time is reached.

The UniqueCode and InstallCode are reference parameters (the values are returned to the client). These values are pulled from the certificate and are used by the client to renew the lease. The reason for using these instead of resending the certificate is to improve performance.

When this function is called, the concurrency server first validates the certificate. It confirms that the certificate's signature is valid and pulls the installation code and uniqueinstallcode from the certificate. In order to validate the signature, the server must have the public key for the certificate. This is stored in the web.config file. For scenarios where you are actually deploying the concurrency server to customer sites, you should find a more secure mechanism to attach the public key (embed in a custom version of the application, or license the concurrency server and embed the public key in the certificate for the concurrency server).

The concurrency server also pulls from the certificate the number of licenses available for this installation code. The concurrentcy server will always use the number from the latest certificate that it sees.

Renew

Parameters:

- InstallKey - string
- UniqueInstall – string
- Returns ConcurrencyResults enumeration

This function is used to renew a lease. It must be called before the time specified in LicenseLifetime parameter of the Allocate function has passed.

In the event that the time expires, or if the server resets for some reason, you'll get a LicenseFreeOrRevoked error. You as the developer, have the option as to how you wish to handle this situation.

Our recommended solution is to call Allocate again to reacquire the lease. The Allocate call can fail if other applications grab the available leases. In this case it is up to you to decide how to proceed. In the example we provide, we allow the application to continue to run, but continue to attempt to reacquire the lease.

Release

- InstallKey - string
- UniqueInstall – string
- Returns ConcurrencyResults enumeration

This function is used to release a lease. It should be called when the application terminates.

ConcurrencyResults Enumeration

The ConcurrencyResults enumeration is used to return results from the concurrency server functions. The possible return values are as follows:

0 – Success	Operation succeeded
1 – LicenseCountExceeded	There are no more leases available on Allocate
2 – LicenseFreedOrRevoked	This is the success result for the Release function. On Renew, this result indicates that the lease expired before it renewed, or that the server was reset.
3 – CertificateInvalid	On Allocate, the certificate passed as the 'cert' property is not valid. It might be corrupt or invalid XML or a signature verification error.
4 – ServerCryptographyError	On Allocate, unable to obtain an RSACryptoServiceProvider on the server. This represents a server configuration error.
5 – AppNotSupported	On Allocate, the application name specified in the certificate does not have a corresponding public key in the configuration server's web.config file.
6 – InvalidKeyForApplication	On Allocate, the public key in the web.config file for this application is not valid.
7 – CertConcurrencyDataError	On Allocate, the concurrency fields are missing from the certificate Server Data, or the server data is missing or has an invalid format.

Configuration

In order to validate certificates, the concurrency server must have the public key for each application that it supports. This is necessary because, without it, a client could substitute any certificate created with the licensing system. The public key is stored under appSettings. The key consists of the name of the application followed by "-pk". The key can be found in the .resx file for the application and simply copied from there to the web.config file.

The UnusedInstallKeyHoldTimeMinutes entry instructs the concurrencyserver how long it should keep the records in memory for a given installation code when it has no leases outstanding.

Sample configuration entries:

```
<appSettings>
   <add key="UnusedInstallKeyHoldTimeMinutes" value="120"/>
```

```
   <add key="mytestapp-pk"
value="&lt;RSAKeyValue&gt;&lt;Modulus&gt;ssjhpvrNmqWaAoO8MGzpIeTXTUarrmNDT6r+Ywle8n
J2eyDB0eTsHu1ljxamKaG9EP6h3vfJHh1mfSNAgQpKca7DCrn1aqwcNDYnJQUIONKxcbZXmDKBCysFZPA6n
M5cHDhoMcsmrgD4CHmpP++vEMld/ve/uz3uDC1PN320dAs=&lt;/Modulus&gt;&lt;Exponent&gt;AQAB
&lt;/Exponent&gt;&lt;/RSAKeyValue&gt;"/>
</appSettings>
```

As mentioned earlier, if you are deploying the certificate server on a customer site, you'll want this information stored securely to prevent it from being modified by the customer.

Security Considerations

The following security issues need to be considered both in design and deployment.

Ensuring the Client Reaches the Correct Server

One of the easiest ways to defeat a concurrency licensing scheme is to have different clients connect to different servers. This implementation addresses this potential vulnerability in a number of ways:

- The concurrency server URL is determined on the DLS server and is embedded into the license certificate (which is digitally signed to prevent modification).
- By keeping the concurrency server lightweight and allowing multiple concurrency servers to exist independently (with each key assigned to one server), it is possible and inexpensive for you to host the concurrency servers rather than host them on the client's network. This makes it possible to use SSL to ensure access to the correct server and encrypt the connection.
- Hosting a concurrency server on a end user's network is possible, but is much less secure because it is always possible for an end user to redirect from one URL to another without the client's knowledge if SSL is not used. Please contact Desaware to discuss possible approaches for this kind of scenario.

Available License Count

The number of licenses available for each installation code is stored inside of each certificate. This does mean that with this implementation it is advisable for clients to upgrade their certificates when the number of licenses available is changed. However, this is not absolutely essential.

Because the concurrency server always uses the information from the certificate with the most recent signature date, depending on usage patterns, the server will tend to have the correct number of licenses.

If this is a concern, it would not be difficult to change the server to contact the licensing server and retrieve the license count for an installation code. That change would, however, make it necessary for the concurrency server to be able to reach the license server – which is not always going to be the case.

The DLSExtensions Component

Application Note #1: Extending the Licensing System, introduced the DLSExtensions component. **Please refer to that application note for an introduction to the component. This section will only cover changes to the component.**

The IkeyExtensions table in the database now includes the ConcurrencyServer and ConcurrencyCount columns that contain the server URL and the number of licenses for the specified installation code. Be sure to download version 2 of the original application note – it incorporates changes to the table name and columns.

The IDBLink Interface

The dblink class implements the IDBLink interface that is defined as follows:

```
Public Interface IDBLink
   Function GetExpirationDate(ByVal InstallationKey As String) As Date
   Sub SetExpirationDate(ByVal InstallationKey As String, ByVal NewDate As Date)
   Function ExtendDemo(ByVal UniqueInstallGUID As String, ByVal NewDate As Date) As String
   Function StripXmlSignature(ByVal xmldocstring As String) As String
   Function GetCurrentDemoExpirationDate(ByVal UniqueInstallGUID As String) As Date
   Sub SetConcurrencyInfo(ByVal InstallationKey As String, ByVal Server As String, ByVal count As Integer)
   Function GetConcurrencyInfo(ByVal InstallationKey As String, ByRef server As String, ByRef count As Integer) As Boolean
End Interface
```

The SetConcurrencyInfo and GetConcurrencyInfo functions are new for this application note.

These functions are used to set and retrieve the concurrency information in the database for a given installation key as shown below.

```
Public Function GetConcurrencyInfo(ByVal InstallationKey As String, ByRef server As String, ByRef count As Integer) As
Boolean Implements IDBLink.GetConcurrencyInfo
      Try
         Dim ds As New DataSet
         ds = GetRowForKey(InstallationKey)
         If ds.Tables(0).Rows.Count = 0 Then Return False ' No expiration
         With ds.Tables(0).Rows(0)
            If IsDBNull(.Item("ConcurrencyServer")) OrElse IsDBNull(.Item("ConcurrencyCount")) Then Return False
            server = CStr(.Item("ConcurrencyServer"))
            count = CInt(.Item("ConcurrencyCount"))
            Return (True)
         End With
      Catch ex As Exception
         Trace.WriteLine("Error in GetConcurrencyInfo: " & ex.Message)
         Throw (ex)
      Finally
         If theconnection.State = ConnectionState.Open Then theconnection.Close()
      End Try

   End Function
```

```
Public Sub SetConcurrencyInfo(ByVal InstallationKey As String, ByVal Server As String, ByVal count As Integer) Implements
IDBLink.SetConcurrencyInfo
    Try
        Dim ds As New DataSet
        ds = GetRowForKey(InstallationKey)
        Dim therow As DataRow
        If ds.Tables(0).Rows.Count = 0 Then
            therow = ds.Tables(0).NewRow()
            ds.Tables(0).Rows.Add(therow)
        Else
            therow = ds.Tables(0).Rows(0)
        End If
        therow("InstallationCode") = InstallationKey
        therow("ConcurrencyServer") = Server
        therow("ConcurrencyCount") = count
        theadapter.Update(ds)

    Catch ex As Exception
        Trace.WriteLine("Error in SetExpirationDate: " & ex.Message)
        Throw (ex)
    Finally
        If theconnection.State = ConnectionState.Open Then theconnection.Close()
    End Try
End Sub
```

The Expirations Example

Application Note #1: Extending the Licensing System, introduced the Expirations example. **Please refer to that application note for an introduction to this example. This section will only cover changes to the application.**

The expirations example has been extended with the new concurrency.aspx page that is used to set and retrieve the concurrency server and license count for each key. As before, the dlsextmanagement.asmx web service has been extended to implement the full IDBLink interface with the new GetConcurrencyInfo and SetConcurrencyInfo methods.

```
Protected Sub cmdGetConcurrencyInfo_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
cmdGetConcurrencyInfo.Click
    Dim m As New dlsextmanagement()
    Try
        If txtKey.Text = "" Then
            lblKeyError.Text = "Invalid key"
            Exit Sub
        End If
        Dim server As String = Nothing, count As Integer
        If Not m.GetConcurrencyInfo(txtKey.Text, server, count) Then
            lblKeyError.Text = "No concurrency info"
```

```
            Else
                txtServer.Text = server
                txtCount.Text = count.ToString
            End If
        Finally
            m.Dispose()
        End Try

    End Sub

    Protected Sub cmdSetConcurrencyInfo_Click(ByVal sender As Object, ByVal e As System.EventArgs) Handles
cmdSetConcurrencyInfo.Click
        Dim m As New dlsextmanagement()
        Try
            If txtKey.Text = "" Then
                lblKeyError.Text = "Invalid key"
                Exit Sub
            End If
            m.SetConcurrencyInfo(txtKey.Text, txtServer.Text, CInt(txtCount.Text))
        Finally
            m.Dispose()
        End Try

    End Sub
```

The Concurrency Server

The concurrency server is designed to operate on a standalone basis and is designed for high performance. In particular, the Renew method is expected to be called most often. All lease data is stored in memory in the application object. The Licenser is the visible web service class and is defined below. All of its methods are synchronized and thread safe.

```
Public Class Licenser
    Inherits System.Web.Services.WebService

    ' Allocate's a license
    <WebMethod()> _
    Public Function AllocateLicense(ByVal cert As String, ByVal LicenseLifetime As Integer, ByRef InstallCode As String, _
    ByRef UniqueCode As String) As ConcurrencyResults
        SyncLock Application
            Dim cm As ConcurrencyManager = GetCM()
            'If Not Diagnostics.Debugger.IsAttached Then
            '    Diagnostics.Debugger.Launch()
            '    Diagnostics.Debugger.Break()
            'End If
            Dim cmr As ConcurrencyResults = cm.ParseCertificate(cert, LicenseLifetime, InstallCode, UniqueCode)
```

```
        If cmr = ConcurrencyResults.Success Then
            Return cm.Allocate(InstallCode, UniqueCode)
        End If
        Return cmr
    End SyncLock

End Function


<WebMethod()> _
Public Function AllocateTest(ByVal LicenseLifetime As Integer, ByRef InstallCode As String, _
ByRef UniqueCode As String) As ConcurrencyResults
    SyncLock Application
        'Dim cm As ConcurrencyManager = GetCM()
        'Return cm.Allocate(InstallCode, UniqueCode)
        Return ConcurrencyResults.FunctionDisabled
    End SyncLock
End Function

<WebMethod()> _
Public Function Renew(ByVal InstallCode As String, ByVal UniqueInstall As String) As ConcurrencyResults
    SyncLock Application
        Dim cm As ConcurrencyManager = GetCM()
        Return cm.Renew(InstallCode, UniqueInstall)
    End SyncLock

End Function

<WebMethod()> _
Public Function Release(ByVal InstallCode As String, ByVal UniqueInstall As String) As ConcurrencyResults
    SyncLock Application
        Dim cm As ConcurrencyManager = GetCM()
        Return cm.Release(InstallCode, UniqueInstall)
    End SyncLock

End Function

Private Function GetCM() As ConcurrencyManager
    Dim cm As ConcurrencyManager = CType(Application.Item("CM"), ConcurrencyManager)
    If cm Is Nothing Then
        cm = New ConcurrencyManager
        Application.Item("CM") = cm
    End If
    Return cm

End Function
```

```
End Class
```

The InstallKeyInfo class

```
Public Class InstallKeyInfo
    Private m_AllocatedLicenseList As New Dictionary(Of String, Date)
    Private m_MaxCount As Integer = 2
    Private m_LicenseLifetime As Integer = 300 ' Lifetime of each license in seconds
    Private m_KeyInfoCertDate As Date ' Date of certificate that sent MaxCount
    Private m_LastAccessTime As Date ' Date/Time of most recent access to this object

    Public Sub SetKeyInfo(ByVal count As Integer, ByVal lifetime As Integer, ByVal certdate As Date)
        m_MaxCount = count
        m_LicenseLifetime = lifetime
        m_KeyInfoCertDate = certdate
        m_LastAccessTime = Now
    End Sub

    Public Sub New(ByVal count As Integer, ByVal lifetime As Integer, ByVal certdate As Date)
        SetKeyInfo(count, lifetime, certdate)
        m_LastAccessTime = Now
    End Sub

    Public Sub New()
        MyBase.New()
    End Sub

    Public ReadOnly Property KeyInfoCertDate() As Date
        Get
            Return m_KeyInfoCertDate
        End Get
    End Property

    Public ReadOnly Property LicenseAvailable() As Boolean
        Get
            ClearOldLicenses()
            Return m_AllocatedLicenseList.Count < m_MaxCount
        End Get
    End Property

    Friend ReadOnly Property LastAccessTime() As Date
        Get
            Return m_LastAccessTime
        End Get
    End Property
```

```
    Public Sub AddOrUpdateUnique(ByVal UniqueInstall As String)
        m_LastAccessTime = Now
        m_AllocatedLicenseList.Item(UniqueInstall) = Now
    End Sub

    Public Function Renew(ByVal UniqueInstall As String) As ConcurrencyResults
        m_LastAccessTime = Now
        If m_AllocatedLicenseList.ContainsKey(UniqueInstall) Then
            m_AllocatedLicenseList.Item(UniqueInstall) = Now
            Return ConcurrencyResults.Success
        Else
            Return ConcurrencyResults.LicenseFreedOrRevoked
        End If
    End Function

    Public Sub RemoveUnique(ByVal UniqueInstall As String)
        m_LastAccessTime = Now
        m_AllocatedLicenseList.Remove(UniqueInstall)
    End Sub

    Private Sub ClearOldLicenses()
        Dim current As Integer = 0
        Do Until current >= m_AllocatedLicenseList.Count
            If Now.Subtract(m_AllocatedLicenseList.ElementAt(current).Value).TotalSeconds > m_LicenseLifetime Then
                ConcurrencyManager.Tracer.TraceInformation("Expiring Unique " &
m_AllocatedLicenseList.ElementAt(current).Key)
                m_AllocatedLicenseList.Remove(m_AllocatedLicenseList.ElementAt(current).Key)
            Else
                current += 1
            End If
        Loop
    End Sub

End Class
```

The InstallKeyInfo object contains information about the leases allocated to a single installation code.

The m_AllocatedLicenseList field contains a dictionary – the keys are the UniqueInstallGUID's of each machine that is holding a lease, the value is the most recent renewal time of the lease.

The m_MaxCount field contains the number of leases available.

The m_LicenseLifetime field is the maximum lifetime of each lease.

The m_LastAccessTime object lists the most recent reference to the object. This field is used to expire the InstallKeyInfo object after sufficient time has passed with no leases outstanding.

```
Public Class InstallKeyInfo
```

```
Private m_AllocatedLicenseList As New Dictionary(Of String, Date)
Private m_MaxCount As Integer = 2
Private m_LicenseLifetime As Integer = 300 ' Lifetime of each license in seconds
Private m_KeyInfoCertDate As Date ' Date of certificate that sent MaxCount
Private m_LastAccessTime As Date ' Date/Time of most recent access to this object

Public Sub SetKeyInfo(ByVal count As Integer, ByVal lifetime As Integer, ByVal certdate As Date)
   m_MaxCount = count
   m_LicenseLifetime = lifetime
   m_KeyInfoCertDate = certdate
   m_LastAccessTime = Now
End Sub

Public Sub New(ByVal count As Integer, ByVal lifetime As Integer, ByVal certdate As Date)
   SetKeyInfo(count, lifetime, certdate)
   m_LastAccessTime = Now
End Sub

Public Sub New()
   MyBase.New()
End Sub

Public ReadOnly Property KeyInfoCertDate() As Date
   Get
      Return m_KeyInfoCertDate
   End Get
End Property

Public ReadOnly Property LicenseAvailable() As Boolean
   Get
      ClearOldLicenses()
      Return m_AllocatedLicenseList.Count < m_MaxCount
   End Get
End Property

Friend ReadOnly Property LastAccessTime() As Date
   Get
      Return m_LastAccessTime
   End Get
End Property

Public Sub AddOrUpdateUnique(ByVal UniqueInstall As String)
   m_LastAccessTime = Now
   m_AllocatedLicenseList.Item(UniqueInstall) = Now
End Sub

Public Function Renew(ByVal UniqueInstall As String) As ConcurrencyResults
```

```
        m_LastAccessTime = Now
        If m_AllocatedLicenseList.ContainsKey(UniqueInstall) Then
            m_AllocatedLicenseList.Item(UniqueInstall) = Now
            Return ConcurrencyResults.Success
        Else
            Return ConcurrencyResults.LicenseFreedOrRevoked
        End If
    End Function

    Public Sub RemoveUnique(ByVal UniqueInstall As String)
        m_LastAccessTime = Now
        m_AllocatedLicenseList.Remove(UniqueInstall)
    End Sub

    Private Sub ClearOldLicenses()
        Dim current As Integer = 0
        Do Until current >= m_AllocatedLicenseList.Count
            If Now.Subtract(m_AllocatedLicenseList.ElementAt(current).Value).TotalSeconds > m_LicenseLifetime Then
                ConcurrencyManager.Tracer.TraceInformation("Expiring Unique " & _
                m_AllocatedLicenseList.ElementAt(current).Key)
                m_AllocatedLicenseList.Remove(m_AllocatedLicenseList.ElementAt(current).Key)
            Else
                current += 1
            End If
        Loop
    End Sub

End Class
```

The ConcurrencyManager class does the bulk of the work and contains methods that are called from the Licenser class.

The Allocate method first checks to see if an InstallKeyInfo object exists for the current installation code. If it does, and a license is available, the lease is granted. If the object does not exist, a new one is created and the lease granted.

```
Public Class ConcurrencyManager

    ' The m_AllocatedLicense list includes a list of installation keys
    ' The entry for each key is a list of unique installs of allocated licenses
    ' Each allocated license has a date - when it was last renewed
    Private m_AllocatedLicense As New Dictionary(Of String, InstallKeyInfo)
    Public Shared Tracer As New Diagnostics.TraceSource("ConcurrencyManagerTrace")

    Friend Function Allocate(ByVal Installkey As String, ByVal UniqueInstall As String) As ConcurrencyResults
        ' Grab the list for the existing key
```

```
    Dim keyinfo As InstallKeyInfo = Nothing
    SyncLock m_AllocatedLicense
      If m_AllocatedLicense.TryGetValue(Installkey, keyinfo) Then
        ' Install key already exists
        If Not keyinfo.LicenseAvailable Then
          Tracer.TraceInformation("License count exceeded key=" & Installkey & " unique=" & UniqueInstall)
          Return ConcurrencyResults.LicenseCountExceeded
        End If
        keyinfo.AddOrUpdateUnique(UniqueInstall)
        Tracer.TraceInformation("License granted key=" & Installkey & " unique=" & UniqueInstall)
      Else
        ' It's a new one - up to the caller to reset default values
        keyinfo = New InstallKeyInfo()
        keyinfo.AddOrUpdateUnique(UniqueInstall)
        m_AllocatedLicense.Add(Installkey, keyinfo)
        Tracer.TraceInformation("License granted key=" & Installkey & " unique=" & UniqueInstall)
      End If
    End SyncLock

    Return ConcurrencyResults.Success


  End Function
```

The renewal is a fast operation. If no InstallKeyInfo object is found for the current installation code, the lease is revoked or not granted. Otherwise the InstallKeyInfo function is called to renew the lease. Renewing the lease also updates the most recent access time.

```
  Public Function Renew(ByVal InstallKey As String, ByVal UniqueInstall As String) As ConcurrencyResults
    ' Grab the list for the existing key
    Dim keyinfo As InstallKeyInfo = Nothing
    ' If license can't be renewed, it means it was timed out, or the application cycled
    ' This is a signal from the client app to reacquire a license
    SyncLock m_AllocatedLicense
      If Not m_AllocatedLicense.TryGetValue(InstallKey, keyinfo) Then
        Tracer.TraceInformation("License revoked key=" & InstallKey & " unique=" & UniqueInstall)
        Return ConcurrencyResults.LicenseFreedOrRevoked
      End If
      Dim results As ConcurrencyResults
      results = keyinfo.Renew(UniqueInstall)
      Tracer.TraceInformation("License renewed key=" & InstallKey & " unique=" & UniqueInstall & " status: " _
     & results.ToString)
      Return results
    End SyncLock

  End Function
```

```
    Public Function Release(ByVal InstallKey As String, ByVal UniqueInstall As String) As ConcurrencyResults
        ' Grab the list for the existing key
        Dim keyinfo As InstallKeyInfo = Nothing
        ' If license can't be renewed, it means it was timed out, or the application cycled
        ' This is a signal from the client app to reacquire a license
        SyncLock m_AllocatedLicense
            If Not m_AllocatedLicense.TryGetValue(InstallKey, keyinfo) Then
                Tracer.TraceInformation("License revoked key=" & InstallKey & " unique=" & UniqueInstall)
                Return ConcurrencyResults.LicenseFreedOrRevoked
            End If
            Tracer.TraceInformation("License released key=" & InstallKey & " unique=" & UniqueInstall)
            keyinfo.RemoveUnique(UniqueInstall)
        End SyncLock
        Return ConcurrencyResults.LicenseFreedOrRevoked
    End Function

    ' Sets concurrency info for an installation key
    ' Only updates if info is newer
    Friend Sub SetConcurrencyInfo(ByVal Installkey As String, ByVal MaxCount As Integer, ByVal LicenseTime As Integer,
ByVal InfoDate As Date)
        Dim keyinfo As InstallKeyInfo = Nothing
        If m_AllocatedLicense.TryGetValue(Installkey, keyinfo) Then
            ' Found one
            If InfoDate > keyinfo.KeyInfoCertDate Then
                ' This cert has a later date than the existing one
                Tracer.TraceInformation("Updating concurrency info. MaxCount=" & MaxCount.ToString & " LicenseTime=" &
LicenseTime.ToString & " InfoDate=" & InfoDate.ToShortDateString)
                keyinfo.SetKeyInfo(MaxCount, LicenseTime, InfoDate)
             End If
        Else
            ' It's a new one
            Tracer.TraceInformation("Setting new concurrency info. MaxCount=" & MaxCount.ToString &  _
           " LicenseTime=" & LicenseTime.ToString & " InfoDate=" & InfoDate.ToShortDateString)
            keyinfo = New InstallKeyInfo(MaxCount, LicenseTime, InfoDate)
            m_AllocatedLicense.Add(Installkey, keyinfo)
        End If
        FlushUnusedCachedKeys()
    End Sub
```

The FlushUnusedCachedKeys function looks for any InstallKeyInfo functions that have had no leases for longer than the time set in the UnusedInstallKeyHoldTimeMinutes setting in the web.config file.

```
    Friend Sub FlushUnusedCachedKeys()
        Static expiredminutes As Integer = CInt(ConfigurationManager.AppSettings.Item("UnusedInstallKeyHoldTimeMinutes"))
        Dim current As Integer = 0
        Do Until current >= m_AllocatedLicense.Count
            If Now.Subtract(m_AllocatedLicense.ElementAt(current).Value.LastAccessTime).TotalMinutes > expiredminutes Then
```

```
            Tracer.TraceInformation("Flushing unused key: " & m_AllocatedLicense.ElementAt(current).Key)
            m_AllocatedLicense.Remove(m_AllocatedLicense.ElementAt(current).Key)
        Else
            current += 1
        End If
    Loop
End Sub
```

The ParseCertificate function is called during allocation to validate the incoming certificate and extract from it the Installation code and the UniqueInstallGUID.

```
Friend Function ParseCertificate(ByVal certificate As String, ByVal LicenseLifetime As Integer, ByRef InstallCode As String,
ByRef UniqueCode As String) As ConcurrencyResults
    Dim xdoc As New XmlDocument
    Try
        xdoc.PreserveWhitespace = False
        xdoc.LoadXml(certificate)
    Catch ex As Exception
        Return ConcurrencyResults.CertificateInvalid
    End Try
    ' Now validate the certificate
    ' Find the application name
    Dim appname As String
    Try
        appname = xdoc.SelectSingleNode("//Application/Name").FirstChild.Value
    Catch ex As Exception
        Return ConcurrencyResults.CertificateInvalid
    End Try

    ' Get the key
    Dim pkxml As String
    Try
        pkxml = ConfigurationManager.AppSettings.Item(appname & "-pk")
    Catch ex As Exception
        Return ConcurrencyResults.AppNotSupported
    End Try

    Dim rsakey As RSACryptoServiceProvider
    Dim params As New CspParameters
    params.Flags = CspProviderFlags.NoPrompt
    Try
        rsakey = New RSACryptoServiceProvider(1024, params)
    Catch ex As Exception
        Try
            params.Flags = params.Flags Or CspProviderFlags.UseMachineKeyStore
            rsakey = New RSACryptoServiceProvider(1024, params)
        Catch ex2 As Exception
```

```
        Return ConcurrencyResults.ServerCryptographyError
    End Try

End Try

' Load the key
Try
    rsakey.FromXmlString(pkxml)
Catch ex As Exception
    rsakey.Clear()
    Return ConcurrencyResults.InvalidKeyForApplication
End Try

' Now validate
Dim XmlSignHolder As SignedXml
Dim sig As XmlElement

Try
    sig = CType(xdoc.GetElementsByTagName("Signature").Item(0), XmlElement)
    If sig Is Nothing Then Return ConcurrencyResults.CertificateInvalid
    XmlSignHolder = New SignedXml(xdoc)
    XmlSignHolder.LoadXml(sig)

    If Not XmlSignHolder.CheckSignature(rsakey) Then
        Return ConcurrencyResults.CertificateInvalid
    End If
Catch ex2 As CryptographicException
    Return ConcurrencyResults.CertificateInvalid
Finally
    rsakey.Clear()
End Try

' Now pull the hashtable
Try
    Dim sv As String = xdoc.GetElementsByTagName("ServerData").Item(0).FirstChild.Value
    Dim b As Byte() = Convert.FromBase64String(sv)
    Dim ms As New IO.MemoryStream(b)
    Dim serverdata As IDictionary = Desaware.DLSextensions.DictionarySerializer.Deserialize(ms)
    InstallCode = CStr(serverdata.Item("InstallationCode"))
    SetConcurrencyInfo(InstallCode, CInt(serverdata.Item("ConcurrencyCount")), LicenseLifetime, _
    CDate(serverdata.Item("ConcurrencyDate")))
    UniqueCode = xdoc.SelectSingleNode("//UniqueInstallation/GUID").FirstChild.Value
Catch ex As Exception
    Return ConcurrencyResults.CertConcurrencyDataError
End Try
```

```
    End Function

End Class
```

Concurrency on the Client

Application Note #1: Extending the Licensing System, introduced a modified version of the HighSecurity example that includes software subscription features. **Please refer to that application note for an introduction to this example. This section will only cover changes to the application.**

The HighSecurity example that comes with this application note includes the features of Application Note #1 (subscription implementation) as well as concurrency support.

To test the functionality, first navigate to the http://localhost/{your virtual directory}/expirations/dlsextedefault.aspx page and the concurrency.aspx pages. Set the expiration value and concurrency information for the installation code you wish to use. You may want to create several client installations in order to test the concurrency – you can use virtual machines to accomplish this.

The server data is extracted from the certificate using the ParseServerData function. This function uses the DictionarySerializer class (described in application note #1) to deserialize the HashTable object.

```
Public Shared Function ParseServerData(ByVal xdoc As Xml.XmlDocument) As IDictionary
    Dim sv As String = xdoc.GetElementsByTagName("ServerData").Item(0).FirstChild.Value
    Dim b As Byte() = Convert.FromBase64String(sv)
    Dim ms As New IO.MemoryStream(b)
    Dim serverdata As IDictionary = Desaware.DLSextensions.DictionarySerializer.Deserialize(ms)
    Return serverdata
End Function
```

The concurrency test is done during the Form load event using the AllocateLicense method of the concurrency server. If a lease is acquired, the application is licensed and a renewal timer is enabled. This application uses a 60 second lease time. The timer polls every 10 seconds, which virtually guarantees a the lease will be renewed well before it expires.

```
If ServerData.Contains("ConcurrencyServer") Then    ' Concurrency is active
    concurrencyserver = New concurrencymanager.Licenser()
    concurrencyserver.Url = CStr(ServerData.Item("ConcurrencyServer"))
    Dim lease As concurrencymanager.ConcurrencyResults = _
    concurrencyserver.AllocateLicense(ClientLicense1.Certificate.OuterXml, 60, _
    m_icode, m_ucode)
    If lease = concurrencymanager.ConcurrencyResults.Success Then
        lblLicensed.Text = "Application is licensed - lease acquired"
        LicenseTimer.Enabled = True
    Else
        lblLicensed.Text = "Lease not acquired: " & lease.ToString
```

```
            End If
        End If
```

Every ten seconds the timer renews the lease. If the lease has been revoked, an attempt is made to reacquire the lease. You as the developer, have the option as to how you wish to handle this situation.

```
    Private Sub LicenseTimer_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
LicenseTimer.Tick
        Static intick As Boolean
        If intick Then Exit Sub
        intick = True
        Try
            Dim result As concurrencymanager.ConcurrencyResults = concurrencyserver.Renew(m_icode, m_ucode)
            If result <> concurrencymanager.ConcurrencyResults.Success Then
                ' Try to reacquire
                ' We reacquire because it's possible for a server to do a restart
                Dim newresult As concurrencymanager.ConcurrencyResults = _
                concurrencyserver.AllocateLicense(ClientLicense1.Certificate.OuterXml, 60, m_icode, m_ucode)
                If newresult <> concurrencymanager.ConcurrencyResults.Success Then
                    If lblLicensed.Text <> "License lease lost" Then
                        MsgBox("License lease lost: " & newresult.ToString)
                        lblLicensed.Text = "License lease lost"
                    End If
                    ' What would you like to do here?
                Else
                    lblLicensed.Text = "Application is licensed - lease acquired"
                End If
                Exit Sub
            End If
        Catch ex As Exception
            ' What would you like to do here?
        Finally
            intick = False
        End Try
    End Sub
```

Conclusion

In an actual deployment, it is likely that you would use the web service interface of the expirations application to automate the process of assigning concurrency information to keys as they are created. A single application could access both this web service and the regular management interface to implement a key management system, or this functionality can be integrated into your existing line of business or customer management system.