

## Features

- SalientProtect provides
  - Support for full and trial licenses
  - Class library for custom implementations
  - Helper classes for quick integration
- Protection for Assemblies
  - Uses Public Key Token authenticate calling assembly
  - License attributes make licensing code simple for end user
- Licensing for Applications
  - Based on locking to a specific machine
  - UI Component for Windows Forms applications

## Installation

SalientProtect can be downloaded as either a ZIP or MSI (Windows Installer) file. In either case, you will be able to choose where the files are installed on your system. To use classes from SalientProtect in your project, you must add a DLL reference to SalientProtect.dll (from bin folder under the installation directory) to your project.

To install the *ApplicationLicenseManager* class as a component, right-click in the “Components” or “Windows Forms” toolbar in Visual Studio and select “Add/Remove Items”. Browse for “SalientProtect.dll” and select it – this will add the component to the component list, already checked. The component will now appear on your toolbar and you can drag it onto the design surface of forms and components in your applications, and the necessary references are added to your project automatically.

## Licensing

The default trial license for SalientProtect will cause a licensing exception to be thrown from time to time when using the library. This does not indicate any problem in your code – it simply reminds you that you don’t have a license to use SalientProtect. Licensing for SalientProtect works by adding an attribute to your application assembly or assemblies, which enables SalientProtect for assemblies signed with your strong name key. Licenses can be obtained through our website: <http://www.SalientPoint.co.uk/>.

## More Information

Details of the classes in SalientProtect can be found in the supplied help file. Further information on all SalientPoint products including SalientProtect can be found on our website: <http://www.SalientPoint.co.uk/>

## Licensing Principles

The licensing functionality in SalientProtect is built around two core classes, *ProtectionKey* and *LicenseBase*.

The *ProtectionKey* class is used to wrap a cryptographic key, along with functions for signing and validating a license. Classes inheriting from *LicenseBase* contain information about the license, along with a digital signature.

The cryptographic key is created by the developer and is unique for each application or assembly. It consists of both public and private parts. The private part of the key **MUST** be kept secret, but the public part of the key is shipped embedded in the application or assembly.

A developer approves a license (e.g. once a customer has paid) by populating the signature field using the private part of the key. This is accomplished using the *Sign* function of the *ProtectionKey* class. License signing may be automated as part of a web site sales process.

An application can verify the authenticity of the license using the public part of the key and the *Verify* function.

## ***SalientProtectManager***

SalientProtect is shipped with a utility called SalientProtectManager that enables you to generate cryptographic keys and licenses, save them as XML, and validate their correctness. Where the following samples use keys and licenses they were generated with this tool. Please refer to the SalientProtectManger documentation for further information.

## **SalientLicenseInfo**

In order to issue licenses manually, you will usually need information from the end user such as their Public Key Token or System Code. To make collecting this information easier you can redistribute the SalientLicenseInfo tool to your customers if you so wish. Please refer to the SalientLicenseInfo documentation for more details on this utility.

## ***ProtectionKey***

The *ProtectionKey* for an application or assembly is usually embedded in the application to enable license verification e.g.:

```
string publicKeyXml = "XML copied from SalientProtectManager";  
ProtectionKey pk = new ProtectionKey( publicKeyXml );
```

It is often more convenient to embed the key in the assembly metadata:

```
[assembly: ProtectionKey(@"XML copied from SalientProtectManager ")]
```

In this case the *SalientProtect* helper classes will usually be able to retrieve it for you automatically. Alternatively it can be retrieved programmatically using a constructor overload that takes the assembly as a parameter:

```
ProtectionKey pk = new ProtectionKey( Assembly.GetExecutingAssembly() );
```

## Licensing Assemblies

Licensing for assemblies uses the *LicenseForAssembly* class and is based on the Public Key Token of the assembly to which the license applies.

The following is a simple example of protected code in an assembly.

```
[assembly: ProtectionKey(@"XML copied from SalientProtectManager ")]

public void ProductDLL_UsefulFunc()
{
    ProtectionKey pk = new ProtectionKey( Assembly.GetExecutingAssembly() );

    Assembly client = Assembly.GetCallingAssembly();

    // Populates license based on client assembly attributes
    LicenseForAssembly lic = new LicenseForAssembly( client );

    if ( pk.Validate( lic ) )
    {
        // License content matches signature
    }
    else
    {
        throw new Exception( "Callers license is not valid" );
    }
    // Do useful stuff here, safe in the knowledge that the caller is licensed
}
```

Using the *AssemblyTrialManager* helper class this can be simplified:

```
[assembly: ProtectionKey(@"XML copied from SalientProtectManager ")]

public void ProductDLL_UsefulFunc()
{
    AssemblyTrialManager.Check( 1 );

    // Do useful stuff here, safe in the knowledge that the caller is licensed
}
```

The *Check* function will throw an exception if the client assembly is not licensed or the license is invalid. *AssemblyTrialManager* can also be used to allow developers to evaluate your assemblies without needing a license. For example, by calling

```
AssemblyTrialManager.Check( 10 );
```

the license check will only be applied (randomly) for about 1 in 10 invocations. This should be enough to allow evaluation of your code, but will prevent production use.

Developers using your assembly can store the license in their own assemblies with a single attribute:

```
[assembly: SalientPoint.SalientProtect.LicenseForAssemblyAttribute(
    "Salient Point Self License" , "Salient Point" ,
    "Signature from SalientProtectManager" , "SalientProtect" )]
```

## Licensing Applications

Licensing for applications uses *LicenseForWindowsSerial*, or *LicenseForWindowsSerialWithExpiry* for time-expiring trial licenses. Licenses are tied to a particular machine using the Windows Serial Number, unique to each Windows installation. This is NOT the same as the product key used to unlock Windows, and is perfectly safe to exchange with others. However, to further protect the user a secure cryptographic hash is used in SalientProtect code. This is referred to in end user documentation as the "System Code", to avoid confusion with the windows license code.

Your application code should be modified to check for a license at an appropriate point:

```
[assembly: ProtectionKey(@"XML copied from SalientProtectManager ")]

private void MainForm_Load( object sender , EventArgs e )
{
    ProtectionKey pk = new ProtectionKey( Assembly.GetExecutingAssembly() );

    LicenseForWindowsSerial lic =
        LicenseForWindowsSerial.ReadFromFile( "app_location\\license.xml" );

    if ( pk.Validate( lic ) )
    {
        // License content matches signature
        if ( lic.WindowsSerialHash ==
            LicenseForWindowsSerial.GetWindowsSerialNumberHash() )
        {
            // License WindowsSerialHash field matches calling assembly
        }
        else
        {
            throw new Exception(
                "License System Code does not match this machine!" );
        }
    }
    else
    {
        throw new Exception( "License is not valid" );
    }
}
```

A practical application will need to provide some better mechanism than an exception to handle missing licenses. Usually this will be a dialog redirecting the user to the application website to purchase a license, or request a trial.

The *ApplicationLicenseManager* component provides a complete solution for end user application licensing. It is a component that can be dragged onto the application design surface, and configured through the property editor. The *RequestLicense* function will check for a valid license, and if none is found it will prompt the user for one, complete with appropriate web links and web service request facility (*LicenseService*).

```
if ( !applicationLicenseManager1.RequestLicense( this ) )
{
    MessageBox.Show(
        this ,
        "No valid license was found, so app is closing" ,
        "app Closing" ,
        MessageBoxButtons.OK ,
        MessageBoxIcon.Error );

    Close();
}
```

In order to leave full flexibility for your application, *ApplicationLicenseManager* uses a dedicated interface to store and retrieve the licence. This interface is implemented by your application, and can use the license *ReadFromFile* and *WriteToFile* functions to store the license in a file. Alternatively, you can store the license in *IsolatedStorage* using your own code to read and write it.

Here is an example, using *SalientFileUtility* to store the license in *Isolated Storage* with *Machine* scope:

```
#region IPersistLicense Implementation

XmlObjectStore< LicenseBase> _xosLicense =
    new XmlObjectStore< LicenseBase>(
        "MyLicense.xml" ,
        Isol.IsolatedStorageScope.Machine |
        Isol.IsolatedStorageScope.Domain |
        Isol.IsolatedStorageScope.Assembly );

LicenseBase IPersistLicense.Read()
{
    try
    {
        return _xosLicense.Read();
    }
    catch ( System.IO.FileNotFoundException )
    {
        return null;
    }
}

void IPersistLicense.Write(LicenseBase license )
{
    _xosLicense.Write( license );
}

#endregion
```

## Implementing *LicenseService*

To implement your own *LicenseService* you will need to create a web service matching the definition in *LicenseService.wsdl* (see documentation directory).

The C# implementation for ASP.NET 2.0 will look like this:

```
[WebService(Namespace = "http://www.salientpoint.co.uk/Services/LicenseService")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class LicenseService : System.Web.Services.WebService
{
    [WebMethod]
    public void SignTrialLicense( ref spProtect.LicenseBase license )
    {
        throw new NotImplementedException();
    }

    [WebMethod]
    public void SignFullLicense(
        ref LicenseBase license ,
        string userName ,
        string Password )
    {
        throw new NotImplementedException();
    }
}
```

The implementation of these functions should apply appropriate business logic for your web retail system, only issuing licenses to users who have paid!

SalientPoint offer hosted e-commerce solutions complete with licensing service. Please contact SalientPoint via the website at <http://www.salientpoint.co.uk> for more information.

## Creating Keys and Licenses

To create and save a new 1024 bit *ProtectionKey*:

```
ProtectionKey pk = new ProtectionKey( 1024 );

pk.Write( "c:\\my_text_key.xml" );
```

To create a new license:

```
LicenseForWindowsSerial license = new LicenseForWindowsSerial();
license.Product = "MyFunkyApp";
license.Name = "Customer";
license.Company = "-";
license.WindowsSerialHash = "System Code provided by purchaser";
```

To sign a license with an existing key and save it:

```
ProtectionKey = ProtectionKey.Read( "c:\\my_text_key.xml" );

pk.Sign( license );

license.WriteToFile( "c:\\my_test_license.txt" )
```