TrueLicense Library Collection: A Java license controller

Contents:

Introduction

The idea of an open source application license controller strikes me as a good topic for a philosophically deep discussion. However, I needed an application license controller, and I didn't want to write my own. So I did a search and found the TLC (TrueLicense Library Collection, **not** "Tender Loving Care"), an application license controller written in Java and made available under the Apache 2.0 license.

The TLC looked as though it would suit my needs, so I rolled my sleeves up and set to work learning about it. These notes describe what I found.

In a nutshell

In the TLC world a license is simply a file that contains encrypted, digitally signed information. The TLC provides the tools to both generate and examine the contents of these license files. It is up to you to decide what you want your application to do with this information.

The model is this: You generate a license certificate file (called the *license key*) and distribute it to the users of your application. They in turn import the license key into their copy of the application. When a license key is imported (installed) into the application, the TLC writes a copy of it into the Java preferences collection. This is the copy of the license key that the TLC tools will work with until the process of installing a license key is followed again.

Note:

Author: Martin Paulo (http://radio.weblogs.com/0130781/)

The use of the Java preferences API does have side effects. For example:

- If the preferences are deleted, then the license key will need to be re-installed.
- If you have two applications storing their license keys in the same node the one will overwrite the other. Normally, not a good thing to happen!
- You need to make sure that your application has permission to read and write to the preferences node that you choose to store the license key in.

To avoid some of these issues you should use the user preference node associated with the main class of your application. If you follow Sun's recommendation to create globally unique package names there shouldn't be clashes with other applications.

With a little extra work this scheme offers you the ability to support separate licenses for different components of your application. So you can create an application that will switch on or off different parts of itself dependent on the licenses that are installed into the application. Now isn't that cool?

When using the TLC you need to provide two Java key stores:

- One to be distributed with your application, containing your public key.
- The other, your personal and private key store, containing your private key.

The file holding the license key is signed with your private key, then encrypted and distributed to your application users. Your public key, in the distributed key store, is used to test the validity of the license key in the file once it has been decrypted.

You should ***never*** distribute the key store containing your private key. TLC enforces this by throwing an exception if a private key is found in the key store entry when verifying a license key. Dare it be suggested that you can make use of this behaviour : add a unit test to your build process that verifies that your private key store is not being shipped to the world!

How to get a copy

For security reasons, the TLC is currently only available as source. You have to compile and build the code yourself.

The first and simplest way is to point your Netbeans IDE at the TLC source repository, and to suck the code out onto your local machine. Build and run the tests to verify that all is correct.

This is all well documented on the TLC home page: https://truelicense.dev.java.net/

For the rest of us that don't normally use Netbeans, life is a little more interesting. I found myself collapsing all of the TLC subprojects into a single project in my Eclipse IDE. However, a customizable build script for Apache's ant is also provided.

A walk through the top level classes

As a user the main classes to be familiar with are:

- The LicenseContent class.

- The LicenseManager class.
- The KeyStoreParam interface.
- The CipherParam interface.
- The LicenseParam interface.

The class LicenseContent is a Java Bean that acts as a thin wrapper around the unencrypted contents of a license, such as information about the issuer of a license or its issue date.

The LicenseManager class allows the using applications to either install, verify or create license certificate files.

An implementation of the KeyStoreParam interface provides the information required to access the key store used to sign or verify signatures.

An implementation of the CipherParam interface provides the pass phrase used when performing the PKCS-5 based encryption and decryption.

An implementation of the LicenseParam interface provides the LicenseManager class with the KeyStoreParam and CipherParam implementations to be used for this particular run of the TLC.

The LicenseManager class will return a LicenceContent instance for the application to work with by means of the verify() method call.

Note:

There is an additional package de.schlichtherle.license.ftp which holds some subclasses and interfaces of these sharing the same names which are to be used for applications which want to support a free trial period. FTP in the TLC code base stands for “Free Trial Period” and **not** for “File Transfer Protocol”!

More classes

The following may help if delving further into the depths of TLC:

The GenericCertificate class is a non-visual JavaBean that wraps the license information and its signature. Its design is inspired by the java.security.SignedObject class, but is enhanced to aid secure usability and long term persistence.

The LicenseNotary class contains the information and smarts required to sign and verify Objects and GenericCertificates.

Note:

In the Preferences the license key is hard wired to the key name “licence”. Again, be sure your different licenses for different products go into different nodes of the Java Preferences.

Working with the TLC

Before working with the TCL create a key store that contains a private and public key pair:

```
keytool -genkey -alias privatekey -keystore privateKeys.store
```

Remember, this key store contains your private key. And as such the world should not be able to get their hands on it. This key store should be kept under lock and key!

From this private key store export the public key to a certificate file:

```
keytool -export -alias privatekey -file certfile.cer -keystore privateKeys.store
```

The certificate file is simply a mechanism for transporting the public key to the key store that will be distributed with the application.

Then import the certificate containing the public key into a new key store that will be distributed with the application:

```
keytool -import -alias publiccert -file certfile.cer -keystore publicCerts.store
```

After this exercise the certificate file can be deleted.

The contents of a license

The class LicenseContent is a non-visual JavaBean containing the following properties:

- The license 'holder', represented as an X.500 principal.
- The license 'issuer', represented as an X.500 principal.
- The artefact being licensed (termed the 'subject'), represented as a String.
- The date the license was 'issued', represented as a Date.
- A date before which the license is not valid (termed 'notBefore'), represented as a Date.
- A date after which the license expires (termed 'notAfter'), represented as a Date.
- The type of license user (called a 'consumerType'), represented as a String.
- The number of licensed users (the 'consumerAmount'), represented as an int.
- Arbitrary information ('info'), represented by a String.

This information is validated by the LicenseManager class whenever a license certificate is created, installed or verified. Validation consists of the following tests:

- The 'subject' must match the subject required by the application via the LicenseParam interface.
- 'holder', 'issuer' and 'issued' must be provided (i.e. not null).
- If 'notBefore' or 'notAfter' are provided, the current date and time must match their restrictions.
- 'consumerType' must be provided and 'consumerAmount' must be positive.
- If the license is stored in a user preference node 'consumerType' must be set to “User” (case is ignored) and the 'consumerAmount' must equal 1.

If you need different license conditions then subclass LicenseManager and override

the method validate(LicenseContent).

How do I create a license?

In the application that generates license files set up an implementation of the KeyStoreParam interface that returns the information required to work with the key store containing the private key:

```
KeyStoreParam privateKeyStoreParam = new KeyStoreParam() {
	public InputStream getStream() throws IOException {
		final String resourceName = "privateKeys.store";
		final InputStream in = getClass().getResourceAsStream(resourceName);
		if (in == null)
			throw new FileNotFoundException(resourceName);
		return in;
	}

	public String getAlias() {
		return "privatekey";
	}

	public String getStorePwd() {
		return "privateKeystorePassword";
	}

	public String getKeyPwd() {
		return "privateKeyPassword";
	}
};
```

Set up an implementation of the CipherParam interface to return the password to be used when performing the PKCS-5 encryption.

```
CipherParam cipherParam = new CipherParam() {
	public String getKeyPwd() {
		return "An arbitrary password";
	}
};
```

Set up an implementation of the LicenseParam interface.

```
LicenseParam licenseParam = new LicenseParam() {
	public String getSubject() {
		return "A revolutionary application";
	}

	public Preferences getPreferences() {
		return Preferences.userNodeForPackage(RunMain.class);
```

```
	}

	public KeyStoreParam getKeyStoreParam() {
		return privateKeyStoreParam;
	}

	public CipherParam getCipherParam() {
		return cipherParam;
	}
};
```

Note that the subject string returned by getSubject() must match the subject property of any LicenseContent instance to be used with this LicenseParam instance.

Set up the LicenseContent instance. This is the information that will be in the generated license file.

```
LicenseContent createLicenseContent() {
	LicenseContent result = new LicenseContent();
	X500Principal holder = new X500Principal("CN=The Using Firm");
	result.setHolder(holder);
	X500Principal issuer = new X500Principal(
			"CN=Blox LLC, L=Wellington, ST=North Island, O=blox,"
				+ " OU=Software Development,"
				+ " C=New Zealand,"
				+ " STREET=28 Elmslie Road Pinehaven, "
				+ " DC=NZ UID=Martin");
	result.setIssuer(issuer);
	result.setConsumerAmount(10000);
	result.setConsumerType("Documents");
	result.setInfo("Limit the number of documents that can be used");
	Date now = new Date();
	result.setIssued(now);
	now.setYear(now.getYear() + 1);
	result.setNotAfter(now);
	result.setSubject(createLicenseParam.getSubject());
	return result;
}
```

Then create the license file!

```
LicenseManager lm = new LicenseManager(licenseParam);
lm.store(createLicenseContent(), new File("sample.lic"));
```

The generated file named “sample.lic” can now be distributed to the appropriate clients.

How do I install a license?

In the application that is distributed to clients set up an implementation of the KeyStoreParam interface that returns the information required to work with the public key store that was distributed with the application:

```
KeyStoreParam publicKeyStoreParam = new KeyStoreParam() {

	public InputStream getStream() throws IOException {
		final String resourceName = "publicCerts.store";
		final InputStream in = getClass().getResourceAsStream(resourceName);
		if (in == null)
			throw new FileNotFoundException(resourceName);
		return in;
	}

	public String getAlias() {
		return "publiccert";
	}

	public String getStorePwd() {
		return "publicCertsStorePassword";
	}

	public String getKeyPwd() {
		// These parameters are not used to create any licenses.
		// Therefore there should never be a private key in the keystore
		// entry. To enforce this policy, we return null here.
		return null; // causes failure if private key is found in this entry
	}
};
```

Set up an implementation of the CypherParam interface to return the same password that that was used when used when performing the PKCS-5 encryption on the license file.

```
CipherParam cipherParam = new CipherParam() {
	public String getKeyPwd() {
		return "An arbitrary password";
	}
};
```

Set up an appropriate implementation of the LicenseParam interface.

```
LicenseParam licenceParam = new LicenseParam() {
	public String getSubject() {
		return "A revolutionary application";
	}
```

```
	public Preferences getPreferences() {
		return Preferences.userNodeForPackage(RunMain.class);
	}

	public KeyStoreParam getKeyStoreParam() {
		return publicKeyStoreParam;
	}

	public CipherParam getCipherParam() {
		return cipherParam;
	}
};
```

Then create the code that will read and install the license file!

```
LicenseManager lm = new LicenseManager(licenceParam);
File licenseFile = new File("sample.lic");
lm.install(licenseFile);
```

From now on the client application simply needs to verify the license information when in it is run.

How do I verify a license?

Using the same KeyStoreParam, CypherParam and LicenseParam implementations that were used to install the license file, one simply writes:

```
LicenseContent lc = lm.verify();
System.out.println(lc.getSubject() +" licensed for use on up to "
+lc.getConsumerAmount() + " " + lc.getConsumerType());
```

Easy, isn't it?

If worried about reverse engineering, place this test in multiple locations in your source code. This should not affect performance too badly as the license content is cached for about half an hour before the verification is started from scratch again.

If you are worried about performance use a background thread to perform the initial verification. From that point on regular calls to the verify() method from your main thread again will then return the cached license content if the verification in the background thread succeeded. The TLC has been developed to be thread safe to support this.

How do I automatically create a free trial period (FTP) license?

The pattern is very similar to the previous examples. You create an implementation of the de.schlichtherle.license.ftp.LicenseParam interface.

Note:

Be careful: This interface extends the regular de.schlichtherle.license.LicenseParam interface, adding information about the free trial period.

Then use an instance of the de.schlichtherle.license.ftp.LicenseManager rather than the regular de.schlichtherle.license.LicenseManager class.

The FTP LicenceManager attempts to verify the standard license first, but if this fails and the user is checked to be eligible for a free trial period, it will create and install a free trial period license automatically. The system will work until such time as the FTP license expires. At this point the user will have to install a regular license.

To guard against recreations of free trial period licenses, the methods “boolean isFTPEligible()” and “void removeFTPEligibility()” are provided in the de.schlichtherle.license.ftp.LicenseParam interface. You should carefully implement these methods storing multiple “cookies” on the computer to protect against fraudulent users.

How do I use the wizard?

The TLC provides a wizard that allows the client application to view or install a license file.

To use the wizard prepare your code as you would for a license verification. But instead of the license verification code, execute the following:

```
LicenseManager lm = new LicenseManager(useLicenceParam);
LicenseWizard lw = new LicenseWizard(lm);
lw.showModalDialog();
```

Obfuscating your code

In order to protect your application against reverse engineering and hence against

circumventing the license verification, you should always obfuscate your code. To see how this is done using the ProGuard obfuscator tool, please check the author's tutorial on the TLC's homepage at http://truelicense.dev.java.net.

Conclusion

The TLC gives you a basic set of tools to manage the licensing of your applications. But you do need to provide the framework in which you will administer, manage and make use of your applications licenses.

Licence