

# **Cinnabar License Manager**

---

# **Cinnabar License Manager**

Copyright © 2003-2010 Cinnabar Systems, LLC. All Rights Reserved.

Cinnabar License Manager, CLM, Cinnabar Canner, and Canner are trademarks of Cinnabar Systems LLC.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries.

---

---

---

---

# Table of Contents

- 1. Introduction ..... 1
  - 1.1. Welcome to Cinnabar License Manager ..... 1
  - 1.2. What is Software Licensing? ..... 1
  - 1.3. What is Runtime License Enforcement? ..... 1
  - 1.4. What is Cinnabar License Manager? ..... 1
  - 1.5. Supported License Conditions ..... 1
- 2. Using CLM ..... 3
  - 2.1. Overview ..... 3
  - 2.2. Creating Your License-Signing Keypair ..... 3
  - 2.3. Creating License Files ..... 4
  - 2.4. Integrating with CLM APIs ..... 7
  - 2.5. Distributing CLM Binary Files ..... 8
  - 2.6. Managing License-Signing Keypairs ..... 8
  - 2.7. Managing your CLM License File ..... 9
- 3. Tool Reference ..... 10
  - 3.1. Using the LicenseTool Program ..... 10
  - 3.2. Using the Ant Tasks ..... 14

---

# Chapter 1. Introduction

## 1.1. Welcome to Cinnabar License Manager

Thanks for choosing Cinnabar License Manager (CLM) to help enforce your software licensing terms! Each software vendor has different requirements for runtime license enforcement. The beauty of CLM is that it is powerful and flexible enough to address almost any software licensing need. This document describes the concepts of software licensing, why you should use runtime license enforcement, and how CLM can enable you to do just that.

## 1.2. What is Software Licensing?

Software Licensing is the process by which most computer users acquire their software. Software is almost always licensed, rather than sold. Thus, when an end user pays for software, what he is actually paying for is not the software itself, but rather the right to use the software. The right of use that is granted to an end user usually comes with certain conditions, specified by you, the licensor. These conditions may restrict the user's right of use to a limited time period, a limited number of machines, or in some other way. In most cases, such conditions are detailed in a printed or on-line license document.

## 1.3. What is Runtime License Enforcement?

Although most licenses purchased by end users come with conditions of use, in most cases the licensing conditions are not enforced in any proactive way. That is, once the user agrees to the conditions, the licensor simply trusts that the user will abide by them. Runtime License Enforcement enables you to enforce the conditions of a license in a proactive way, every time the software is used. If the license conditions are violated, your software can take some appropriate action, as defined by you.

## 1.4. What is Cinnabar License Manager?

CLM is a class library that enables programs written in the Java language to use Runtime License Enforcement. The CLM client API is very simple, with a single function that checks to ensure that the current runtime environment is in compliance with the desired licensing parameters. To illustrate the simplicity of the CLM API, consider the following code fragment:

```
LicenseManager mgr = new LicenseManager(clmLicenseStream,  
    productLicenseStream, publicKeyStream);  
  
mgr.acquireLicense(product, feature);
```

In this example, if the call to `acquireLicense` returns without throwing an exception, then the current runtime environment is in compliance with the license conditions specified by the license file. It's that simple!

## 1.5. Supported License Conditions

CLM uses the following license parameters to ensure runtime compliance with the license conditions. Using these parameters, you can define what you are licensing, who is authorized to use it, and when or where they are authorized to use it.

## Product Name

The `Product Name` parameter identifies the product being licensed. This ensures that licenses issued for one product cannot be used for a different product.

## Feature Name

The `Feature Name` parameter identifies a particular feature being licensed. Product features can be specified individually, allowing you to enable product features dynamically at runtime, based on the contents of the license file. This enables you to sell your products on a feature-by-feature basis, and upgrade users in the field with enhanced functionality by simply shipping them a new license file.

## Client Ethernet MAC Address

The `Client Ethernet MAC Address` parameter allows you to restrict use of your product to a machine with the specified MAC address. MAC addresses are 48 bits long, and every Ethernet network card in the world has a unique value. This makes the MAC address a useful way to identify a particular machine.

## Client Domain Name

The `Client Domain Name` parameter allows you to restrict use of your product to a machine that is located in a particular domain. For example, you might wish to license your product only for use inside XYZ Corporation. You could use the `Client Domain Name` parameter to specify that your product should only run if the hostname of the machine ended in "xyzcorp.com".

## Expiration Date

The `Expiration Date` parameter allows you to restrict use of each feature to a specified time period. This allows you to sell time-limited licenses, as well as distribute free demonstration licenses that offer full or restricted product functionality for a limited time. After the specified expiration date has been reached, the product can stop functioning, or function at a reduced feature level.

## Days Since First Use

The `Days Since First Use` parameter allows you to restrict use of each feature to a specified number of days since the feature was first used. This allows you to sell time-limited licenses, as well as distribute free demonstration licenses that offer full or restricted product functionality for a limited time. After the specified number of days have elapsed since the first use of the feature, the product can stop functioning, or function at a reduced feature level.

---

# Chapter 2. Using CLM

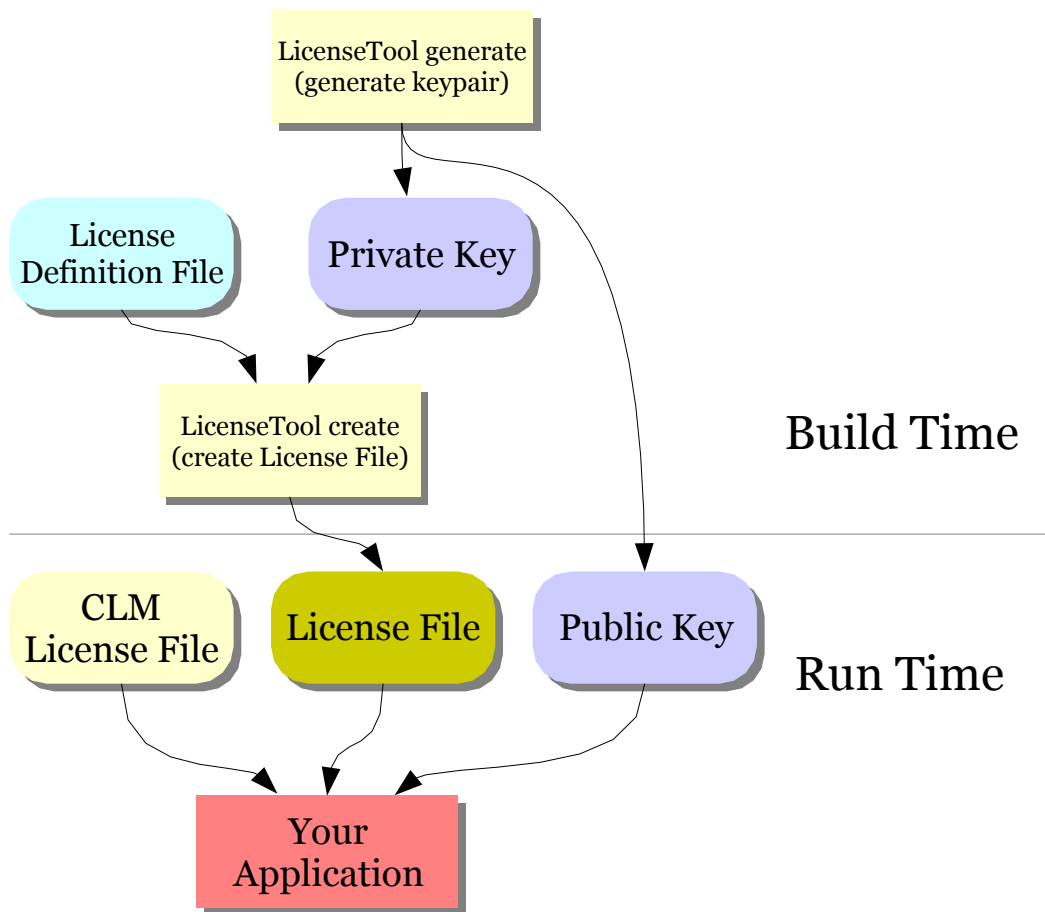
## 2.1. Overview

In order to use Cinnabar License Manager to do runtime license enforcement, you must perform the following steps:

1. Create your license-signing keypair
2. Create license files for your application
3. Integrate your application with the CLM APIs
4. Distribute CLM binary files with your application

The following illustration gives an overview of the entire licensing process.

**Figure 2.1. CLM Overview**



## 2.2. Creating Your License-Signing Keypair

CLM uses public-key cryptography to ensure the integrity of license files in the field (more about license files in a moment). Since CLM uses public-key cryptography, you have to create a keypair. A keypair is a matched set of keys – one private key, and one public key.

To create a keypair, you can use the **LicenseTool** program that comes with CLM (see Section 3.1, “Using the LicenseTool Program” [10] for details). You can also use the **generateKeypair** Ant task to generate a keypair using an Ant build file (see Section 3.2, “Using the Ant Tasks” [14] for details). Once you create a keypair, you use the private key to sign license files, and the public key to verify license files. You can decide if you want to use a given keypair for all the license files your company generates, or for only the license files generated for a particular product, or even for just a single use. The frequency with which you generate a new keypair is up to you.

## 2.3. Creating License Files

To create a license file, you first specify the conditions of your license in a License Definition File (LDF). Then, you use your private key to sign the LDF, using the **LicenseTool** program. The LDF and its signature are packaged into a single file called the License File.

### Creating License Definition Files

An LDF is a simple XML document. The DTD file `LicenseDefinitionFile.dtd` is provided with CLM for your convenience. The syntax of an LDF is described below.

#### License Definition File Syntax

##### The license Element

Each LDF has a top-level element called `license`. The `license` start-tag must have a `name` attribute, which specifies the name of the product being licensed.

For example, the following `license` start-tag specifies that the name of the product being licensed is "WidgetMaster6000":

```
<license name="WidgetMaster6000">
```

##### The licensee Element

Each `license` element contains one or more child `licensee` elements. Each `licensee` start-tag must have a `name` attribute, which specifies the name of the party to whom this `licensee` element pertains.

For example, the following `licensee` start-tag specifies that this license is for "Ralph's Stop-n-Shop":

```
<licensee name="Ralph's Stop-n-Shop">
```

You may use a generic or empty licensee name if you intend to reuse the generated License File for multiple customers.

##### The feature Element

Each `licensee` element contains one or more child `feature` elements. Each `feature` start-tag must have a `name` attribute, which specifies the name of the feature to which this `feature` element pertains.

For example, the following `feature` start-tag specifies that this feature is called "saveToXml":

```
<feature name="saveToXml">
```

## The attribute Element

The `license`, `licensee`, and `feature` elements may all contain zero or more child `attribute` elements. All `attribute` child elements, if any, must appear before any other child elements.

Each `attribute` start-tag must contain three attributes, `name`, `type`, and `value`. The `name` attribute specifies the name of this `attribute`. The `type` attribute specifies the type of this `attribute`, and must have the value `int` or `string`. The `value` attribute specifies the value of the `attribute`. For `attributes` of type `int`, the `value` must specify an integer. For `attributes` of type `string`, the `value` can be any string.

For example, the following `attribute` element describes a string `attribute` named `issuedBy`, with the value `Nancy`:

```
<attribute name="issuedBy" type="string" value="Nancy"/>
```

## Predefined Attributes

CLM recognizes several predefined `attribute` names. If an LDF contains any `attributes` with these predefined names, CLM interprets their values in a special way. The names of all the predefined `attributes` recognized by CLM begin with `clm-`. The prefix `clm-` for `attribute` names is reserved for use by CLM predefined `attributes`.

## License Attributes

CLM recognizes no predefined `attribute` names in the `license` element.

## Licensee Attributes

CLM recognizes the following predefined `attribute` names in the `licensee` element:

Name	Type	Value Interpretation	Example
<code>clm-NodeId</code>	<code>string</code>	An Ethernet MAC address. Must be exactly 12 characters long, and composed only of the characters 0-9, a-f, and A-F	0020781F4336
<code>clm-Domain</code>	<code>string</code>	A domain name suffix	example.com

## Feature Attributes

CLM recognizes the following predefined attribute names in the feature element:

Name	Type	Value Interpretation	Example
clm-ExpireDate	string	A date specification. Must be exactly 8 characters long, in the format "yyyyMMdd".	20030301
clm-ExpireDays	int	The number of days to allow use of the enclosing Feature, after the first use.	30

## Unrecognized Attribute Names

All attributes present in an LDF are available via the CLM APIs at runtime. Thus, you can add your own custom attributes in the LDF, and use them at runtime to vary the behavior of your program.

## Sample License Definition Files

### Fixed Expiration License

The following LDF specifies that the feature "saveToXml" of the product "WidgetMaster6000" is licensed to operate on any machine until 23:59:59 on 31 May 2003. The license was issued to "Ralph's Stop-n-Shop".

```
<license name="WidgetMaster6000">
  <licensee name="Ralph's Stop-n-Shop">
    <feature name="saveToXml">
      <attribute name="clm-ExpireDate" type="string"
        value="20030601"/>
    </feature>
  </licensee>
</license>
```

### Relative Expiration License

The following LDF specifies that the feature "saveToXml" of the product "WidgetMaster6000" is licensed to operate on any machine for a period of 30 days after its first use. The license was issued to "Ralph's Stop-n-Shop".

```
<license name="WidgetMaster6000">
  <licensee name="Ralph's Stop-n-Shop">
    <feature name="saveToXml">
      <attribute name="clm-ExpireDays" type="int" value="30"/>
    </feature>
  </licensee>
</license>
```

## Node-locked, Non-expiring License

The following LDF specifies that the feature "saveToXml" of the product "WidgetMaster6000" is licensed to operate on a machine with the MAC address 0020781F4336, in perpetuity. The license was issued to "Ralph's Stop-n-Shop". This license file also contains a custom attribute in the license element.

```
<license name="WidgetMaster6000">
  <attribute name="issuedBy" type="string" value="Nancy"/>
  <licensee name="Ralph's Stop-n-Shop">
    <attribute name="clm-NodeId" type="string"
      value="0020781F4336"/>
    <feature name="saveToXml"/>
  </licensee>
</license>
```

## Signing License Definition Files

After you create an LDF, you must sign it with your private key to create a License File. To sign an LDF, you can use the **LicenseTool** program that comes with CLM (see Section 3.1, "Using the LicenseTool Program" [10] for details). You can also use the **createLicense** Ant task to create a new license file using an Ant build file (see Section 3.2, "Using the Ant Tasks" [14] for details).

## 2.4. Integrating with CLM APIs

Integrating your application with CLM is very easy. You simply create a `LicenseManager` object, and then call the `acquireLicense` method of that object to verify that a license is available for the specified product and feature. To create a `LicenseManager` object, instantiate an object of the class `com.cinnabarsystems.clm.LicenseManager`, specifying the source of your CLM license file (provided to you by Cinnabar Systems), the source of the product license file(s) to check, and the public key to use to verify the product license files. Several constructors are available to accommodate different sources for the license files and public key data. For example, the following code creates a `LicenseManager` object that reads the CLM and product licenses from `FileInputStreams`, and the public key data from a resource stream:

```
InputStream myCLMLicenseStream =
    new FileInputStream("clm.lic");
InputStream myLicenseStream =
    new FileInputStream("license.lic");
InputStream myPublicKeyStream =
    getClass().getResourceAsStream("pubkey");
LicenseManager mgr = new LicenseManager(myCLMLicenseStream,
    myLicenseStream, myPublicKeyStream);
```

There are other convenient `LicenseManager` constructors that can read all the license files in a given directory – see the online API documentation for full details.

Once you create the `LicenseManager` object, you can call the `acquireLicense` method to attempt to acquire a license for a given product and feature. If the license acquisition is successful (i.e., at least one of

the specified license files grants a license for that product and feature in the current execution environment), then `acquireLicense` will return successfully. If the license acquisition is not successful, then `acquireLicense` will throw some subclass of `LicenseManagerException`.

```
try
{
    License license = mgr.acquireLicense(product, feature);
}
catch(LicenseManagerException lme)
{
    // A license could not be acquired; take some
    // licenser-defined action
}
```

If `acquireLicense` returns successfully, it returns a `License` object. This `License` object is the root of an object hierarchy representing the license parameters in the license file that satisfied the license acquisition request. You can use the `License` object to obtain more detailed information about the license file that satisfied the acquisition request.

```
String issuedBy = (String)license.getAttribute("issuedBy");
```

## 2.5. Distributing CLM Binary Files

After your application is integrated with the CLM APIs, it will require CLM code to run. The following table summarizes the CLM binary files you must distribute with your application.

Filename	Required Runtime Location
<code>clm.jar</code>	Anywhere in the JVM classpath
<code>cinnabar.dll</code>	In the same directory as <code>clm.jar</code> , or anywhere in the system PATH

If you are using Cinnabar Systems' Canner™ to distribute your application, then the files listed above will be bundled into your executable, and it is not necessary to distribute them separately. You can find more information about Canner™ at <http://www.cinnabarsystems.com>.

## 2.6. Managing License-Signing Keypairs

When you create a License File, you do so using a License Definition File and the private key of a keypair. Later, your application uses the public key of the same keypair to verify the License File. Every License File is specific to a keypair – it can only be verified by the public key that is matched to the private key that you used to create the License File. Managing the storage and distribution of the keys in your license-signing keypair is critical to assure the effectiveness and security of CLM.

The private key of a keypair should never be distributed outside of your organization. It should be treated as a business asset, and safeguarded to prevent outside distribution.

The public key of a keypair must be accessible to your application at runtime, so that CLM can use it to

verify that the License File is genuine and unmodified. Your application passes an `InputStream` for accessing the public key to the constructor of its `LicenseManager` object. Possible means of storing the public key are:

- As a resource in your application's jar file
- As a static byte [] in your application's source code
- As static data in a native code library, accessed by JNI
- As a distant resource accessed over the network

The security of the public key is important – if an attacker can change the public key, he can create and sign his own License Files, thus defeating CLM. Unfortunately, the nature of Java makes it difficult to ensure secure storage of any Java code or resource. Making it reasonably difficult for an attacker to change the public key, combined with obfuscating your application's class files, will discourage most licensees from investing time in defeating your application's runtime license enforcement code.

To enhance the security of public key storage, as well as to make decompilation of your Java programs all but impossible, we suggest you use Canner™, a separate product of Cinnabar Systems. You can find more information about Canner™ at <http://www.cinnabarsystems.com>.

## 2.7. Managing your CLM License File

Cinnabar Systems provided you with a license file that authorizes you to use CLM. You must distribute this license file with your applications that use CLM, and you must provide an `InputStream` to this license file as the first parameter to all the `LicenseManager` constructors.

Cinnabar Systems recommends that, when you distribute your application, you bundle the CLM license file into your application jarfiles- this prevents you from having to distribute an additional file with your applications. Bundling the CLM license file into your application jarfile also simplifies the process of protecting your applications with Cinnabar Canner- the CLM license file will be encrypted and included in the canned executable file, along with the Java classes in your jarfile.

---

# Chapter 3. Tool Reference

## 3.1. Using the LicenseTool Program

**LicenseTool** is a utility program to help you create and test License Files, as well as create new license-signing keypairs. To run **LicenseTool**, use the `java -jar` command to execute the jarfile `clm-tools.jar`. Note that the file `cinncabar.dll` must be in the system PATH or in the same directory as `clm.jar` when **LicenseTool** is executed.

**LicenseTool** returns a zero value to the command shell to indicate success, and a non-zero value to indicate failure.

### Generating a New License-Signing Keypair

**LicenseTool** can be used to generate a new license-signing keypair. To generate a new license-signing keypair, specify the following arguments to **LicenseTool**:

Argument	Description
<code>generate</code>	Indicates that you want to generate a new license-signing keypair.
<code>-n keyfile</code>	Specifies that the new private key should be written to <code>keyfile.private</code> , and the new public key should be written to <code>keyfile.public</code> .

### Example

The following usage of **LicenseTool** creates a new license-signing keypair. The public key is stored in the file `newkey.public`, and the private key is stored in the file `newkey.private`:

```
java -jar clm-tools.jar generate -n newkey
```

Upon successful completion of a `generate` command, **LicenseTool** produces output like the following:

```
Saved new keys to newkey.private and newkey.public
```

### Creating New License Files

**LicenseTool** can be used to create a new License File. To create a new License File, specify the following arguments to **LicenseTool**:

Argument	Description
<code>create</code>	Indicates that you want to create a new License File.
<code>-i licenseDefinitionFileName</code>	Specifies the input License Definition File.

Argument	Description
-o licenseFileName	Specifies the output License File.
-k privateKeyfile	Specifies an existing private key to be used to sign the License Definition File.
-n keyfile	Specifies that a new license-signing keypair should be created and used to sign the License Definition File. The new private key is written to keyfile.private, and the new public key is written to keyfile.public.

Only one of -k or -n may be specified.

## Examples

The following usage of **LicenseTool** creates a new License File named `mylicense.lic` from the License Definition File `myldf.xml`. **LicenseTool** generates a new keypair in the process – the private key of the new keypair is written to the file `mykey.private`, and the public key of the new keypair is written to the file `mykey.public`.

```
java -jar clm-tools.jar create -i myldf.xml -o mylicense.lic
-n mykey
```

The following usage of **LicenseTool** creates a new License File named `mylicense.lic` from the License Definition File `myldf.xml`. **LicenseTool** uses the existing private key in the file `mykey.private` to generate the License File.

```
java -jar clm-tools.jar create -i myldf.xml -o mylicense.lic
-k mykey.private
```

Upon successful completion of a `create` command, **LicenseTool** produces output like the following:

```
Created license file mylicense.lic
```

## Verifying Existing License Files

**LicenseTool** can be used to verify an existing License File. Verification consists of checking the License File to make sure that the signature is valid, and that the License File has not been altered since it was signed. To verify an existing license, specify the following arguments to **LicenseTool**:

Argument	Description
<code>verify</code>	Indicates that you want to verify an existing License File.
-i licenseFileName	Specifies the input License File.
-k publicKeyfile	Specifies an existing public key to be used to verify the License File. The License File must have been

Argument	Description
	created with the private key that matches the specified public key.
-l clmLicenseFile	Specifies the location of the CLM license file

## Example

The following usage of **LicenseTool** verifies the existing License File named `mylicense.lic`. **LicenseTool** uses the existing public key in the file `mykey.public` to verify the License File. The CLM license file is located in the file `clm.lic`.

```
java -jar clm-tools.jar verify -i mylicense.lic -k mykey.public
-l clm.lic
```

Upon successful completion of a `verify` command, **LicenseTool** produces output like the following:

```
License file mylicense.lic verified ok.
```

If the `verify` command fails for some reason, a descriptive stack trace will be printed.

## Testing License Acquisition

**LicenseTool** can be used to test license acquisition using an existing License File. To test license acquisition, specify the following arguments to **LicenseTool**:

Argument	Description
<code>test</code>	Indicates that you want to test license acquisition.
<code>-p productName</code>	Specifies the Product Name to use when acquiring a license.
<code>-f featureName</code>	Specifies the Feature Name to use when acquiring a license.
<code>-k publicKeyfile</code>	Specifies an existing public key to be used when acquiring the license. The License File must have been created with the private key that matches the specified public key.
<code>-r licenseDirectory</code>	If this argument is present, it specifies a directory from which to read license files, and license acquisition is attempted using all the License Files from that directory. If this argument is not present, license acquisition is attempted using only a single License File.
<code>-i licenseFileNameOrExtension</code>	If the <code>-r</code> argument is present, this argument specifies the file extension for License Files. If the <code>-r</code> argument is not present, this argument specifies the single License File to use.

Argument	Description
-l clmLicenseFile	Specifies the location of the CLM license file

## Examples

The following usage of **LicenseTool** attempts to acquire a license for the Product "WidgetMaster6000" and the Feature "saveToXml", using the existing License File named `mylicense.lic`. **LicenseTool** uses the existing public key in the file `mykey.public` to read the License File. The CLM license file is located in the file `clm.lic`.

```
java -jar clm-tools.jar test -p WidgetMaster6000 -f saveToXml
-i mylicense.lic -k mykey.public -l clm.lic
```

The following usage of **LicenseTool** attempts to acquire a license for the Product "WidgetMaster6000" and the Feature "saveToXml", using all the License Files named `*.lic`. **LicenseTool** uses the existing public key in the file `mykey.public` to read the License File. The CLM license file is located in the file `clm.lic`.

```
java -jar clm-tools.jar test -p WidgetMaster6000 -f saveToXml
-r -i *.lic -k mykey.public -l clm.lic
```

Upon successful completion of a `test` command, **LicenseTool** produces output like the following:

```
License for WidgetMaster6000/saveToXml successfully acquired from
mylicense.lic
```

If the `test` command fails for some reason, a descriptive stack trace will be printed.

## Dumping License Specifications

**LicenseTool** can be used to dump the specifications in an existing License File. To dump a license specification, specify the following arguments to **LicenseTool**:

Argument	Description
dump	Indicates that you want to dump the license specifications from an existing License File.
-i licenseFileName	Specifies the input License File.

## Example

The following usage of **LicenseTool** dumps the license specifications from the existing License File named `mylicense.lic`:

```
java -jar clm-tools.jar dump -i mylicense.lic
```

Upon successful completion of a dump command, **LicenseTool** produces output like the following:

```
license, product name=WidgetMaster6000
  licensee, name=Ralph's Stop-n-Shop
    feature, name=saveToXml
      attribute, name=clm-Expire-Days, value=30
```

## 3.2. Using the Ant Tasks

If you use Apache Ant in your build process, you can easily integrate CLM tools into your build. Five Ant Tasks are provided in the file `clm-ant.jar`. These Tasks provide all the functionality that the **LicenseTool** program provides, in a form that can be easily integrated into an Ant build process.

### Defining the Tasks

Before you can use the Ant Tasks, they must be explicitly defined in the Ant build file. To define the tasks, use the following declarations:

```
<property
  name="clm.classpath"
  value="clm-ant.jar;clm.jar"/>

<taskdef
  resource="com/cinnabarsystems/clm/ant/taskdefs"
  classpath="{clm.classpath}"/>
```

Adjust the definition of the property `clm.classpath` to the appropriate directory for your build environment.

The taskdef declaration shown above defines five new tasks: **generateKeypair**, **createLicense**, **verifyLicense**, **acquireLicense**, and **dumpLicense**. The usage of each of these new tasks is detailed below.

### Generating a New License-Signing Keypair

To generate a new license-signing keypair, use the **generateKeypair** task. The **generateKeypair** task supports the following attributes:

Attribute Name	Definition
<code>newPublicKeyFile</code>	The name of the file in which to save the new public key
<code>newPrivateKeyFile</code>	The name of the file in which to save the new private key

### Example

The following creates a new license-signing keypair. The public key is stored in the file `newkey.public`, and the private key is stored in the file `newkey.private`.

```
<generateKeypair
  newPrivateKeyFile="newkey.private"
  newPublicKeyFile="newkey.public"/>
```

## Creating New License Files

To create a new License File, use the **createLicense** task. The **createLicense** task supports the following attributes:

Attribute Name	Definition
<code>licenseDefinitionFile</code>	The name of the input License Definition File
<code>licenseFile</code>	The name of the output License File
<code>existingPrivateKeyFile</code>	The name of an existing private key file to be used to sign the License Definition File
<code>newPublicKeyFile</code>	The name of the file in which to save the new public key
<code>newPrivateKeyFile</code>	The name of the file in which to save the new private key

If `existingPrivateKeyFile` is present, the named key file is used to sign the License Definition File. If `existingPrivateKeyFile` is not present, then both `newPublicKeyFile` and `newPrivateKeyFile` must be present.

## Examples

The following task creates a new License File named `mylicense.lic` from the License Definition File `myldf.xml`. A new keypair is generated in the process – the private key of the new keypair is written to the file `mykey.private`, and the public key of the new keypair is written to the file `mykey.public`.

```
<createLicense
  licenseDefinitionFile="myldf.xml"
  licenseFile="mylicense.lic"
  newPrivateKeyFile="mykey.private"
  newPublicKeyFile="mykey.public"/>
```

The following task creates a new License File named `mylicense.lic` from the License Definition File `myldf.xml`. The existing private key in the file `mykey.private` is used to generate the License File.

```
<createLicense
  licenseDefinitionFile="myldf.xml"
  licenseFile="mylicense.lic"
  existingPrivateKeyFile="mykey.private"/>
```

## Verifying Existing License Files

To verify an existing License File, use the **verifyLicense** task. The **verifyLicense** task supports the following attributes:

Attribute Name	Definition
licenseFile	The name of the input License File
publicKeyFile	The name of an existing public key file to be used to verify the License File
clmLicenseFile	The name of your CLM license file

### Example

The following task verifies the existing License File named `mylicense.lic`. The existing public key in the file `mykey.public` is used to verify the License File. The CLM license file is located in the file `clm.lic`.

```
<verifyLicense
  licenseFile="mylicense.lic"
  publicKeyFile="mykey.public"
  clmLicenseFile="clm.lic"/>
```

## Testing License Acquisition

To test license acquisition, use the **acquireLicense** task. The **acquireLicense** task supports the following attributes:

Attribute Name	Definition
licenseFile	The name of the input License File
licenseDirectory	The name of a directory containing License Files
licenseFileExtension	The file extension of the License Files
product	The Product name to use when acquiring a license
feature	The Feature name to use when acquiring a license
publicKeyFile	The name of an existing public key file to be used to when acquiring the license. The License File must have been created with the private key that matches the specified public key.
clmLicenseFile	The name of your CLM license file

### Examples

The following task attempts to acquire a license for the Product "WidgetMaster6000" and the Feature "saveToXml", using the existing License File named `mylicense.lic`. The existing public key in the file `mykey.public` is used to read the License File. The CLM license file is located in the file `clm.lic`.

```
<acquireLicense
  licenseFile="mylicense.lic"
  product="WidgetMaster6000"
  feature="saveToXml"
  publicKeyFile="mykey.public"
  clmLicenseFile="clm.lic"/>
```

The following task attempts to acquire a license for the Product "WidgetMaster6000" and the Feature "saveToXml", using all the License Files named \*.lic in the directory named "licenses". The existing public key in the file mykey.public is used to read the License File. The CLM license file is located in the file clm.lic.

```
<acquireLicense
  licenseDirectory="licenses"
  licenseFileExtension=".lic"
  product="WidgetMaster6000"
  feature="saveToXml"
  publicKeyFile="mykey.public"
  clmLicenseFile="clm.lic"/>
```

## Dumping License Specifications

To dump the specifications of an existing License File, use the **dumpLicense** task.. The **dumpLicense** task supports the following attributes:

Attribute Name	Definition
licenseFile	The name of the input License File

### Example

The following task dumps the license specifications from the existing License File named mylicense.lic.

```
<dumpLicense
  licenseFile="mylicense.lic"/>
```