

# **Zen License Manager Documentation**

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Installation</b>	<b>1</b>
<b>3</b>	<b>Installation Windows (Visual Studio)</b>	<b>4</b>
<b>4</b>	<b>ZLM Java bindings</b>	<b>7</b>
<b>5</b>	<b>ZLM Tools</b>	<b>8</b>
<b>6</b>	<b>License File Format</b>	<b>9</b>
<b>7</b>	<b>Integrating ZLM</b>	<b>12</b>
<b>8</b>	<b>API Documentation</b>	<b>13</b>
<b>9</b>	<b>More API Examples</b>	<b>22</b>
<b>10</b>	<b>Extending ZLM</b>	<b>26</b>
<b>11</b>	<b>Setting up MinGW</b>	<b>27</b>
<b>12</b>	<b>Android Specifics</b>	<b>28</b>
<b>13</b>	<b>iOS Specifics</b>	<b>29</b>
<b>14</b>	<b>Tool Sources</b>	<b>30</b>
<b>15</b>	<b>Changelog</b>	<b>33</b>

---

This documentation describes the *Zen License Manager* (ZLM) version 1.23 — a simple and easy-to-use C library and toolset to manage software licenses. It should give you all the information you need to integrate ZLM into your software product and manage the corresponding software licenses. Out of the box ZLM can be linked against C/C++ programs. For most Platforms Java bindings are also contained in the distribution (see Section 4). For Windows (Visual Studio) .NET bindings are provided (see [Using the .NET bindings](#)). With the right language bindings ZLM can be used with pretty much any programming language. [ZLM Go bindings](#) and experimental [ZLM Fortran bindings](#) are available online, please contact us if you need bindings for other programming languages.

## 1 Introduction

To protect your software with ZLM you need to do two simple steps:

1. Add ZLM to your software to protect it with license files (or license strings).
2. Give the licenses to your customers.

Currently there are two “dimensions” in which you can restrict your licenses:

- Site licenses vs. fixed-node licenses.
- Permanent licenses vs. expiring licenses.

A site-license can run on any host computer, a fixed-node license only on a defined set of host computers. A permanent license does never expire, an expiring license is only valid until a certain date.

The licensing strategy is entirely defined by the license files (see Section 6), the code in your program is always the same and defines only from which location the license is loaded.

But more on that later, at first we have to install ZLM.

## 2 Installation

This section describes the installation for *all* platforms *except* Windows (Visual Studio), the installation guidelines for Windows (Visual Studio) are given in Section 3. The installation for Windows (MinGW) is the same as described here, please refer to Section 11 on how to setup MinGW on Windows. The installations for Android and iOS have some peculiarities, please also refer to Section 12 and Section 13 for details.

Perform the following tasks in a shell or terminal, on Windows (MinGW) use the MinGW command line. After you downloaded your ZLM distribution for the desired platform from [ZLM Download](#) you have to extract the archive file and change into the resulting directory. For example like this:

```
tar xzf zlm-1.0-Linux_i686-32bit.tar.gz
cd zlm-1.0-Linux_i686-32bit
```

The distribution contains the following files.

Table 1: The files contained in the distribution

api_example.c	example program which shows how to use the ZLM API
CHANGELOG	the change log
example.lic	example license file
Makefile	make file to build the ZLM tools and the example program
README	a README file containing basic infos
zlm.a	the ZLM library archive
zlmdoc.html	a HTML file containing the ZLM documentation
zlmdoc.pdf	the same documentation as a PDF file

Table 1: (continued)

zlmgenkeys	the executable to generate the public/private key pair
zlmgenkeys.c	the corresponding source file
zlm.h	the ZLM library header file
zlmhostid	the executable to print/write host IDs
zlmhostid.c	the source file for the zlmhostid tool
zlmkeyhash.c	the source file for the zlmkeyhash tool
zlm_license_dummy.c	needed to build the zlmkeyhash tool
zlm_sign.c	the source file for the zlm_sign tool
zlmverify.c	the source file for the zlmverify tool

Now you have to perform three easy steps to set up ZLM:

**Generate public/private key pair** Execute `zlmgenkeys` in this directory to generate a public/private key pair:

```
./zlmgenkeys
```

The output of `zlmgenkeys` looks like this:

```
generating keys...
done!
private key file 'zlm_privkey.c' was written
public key file 'zlm_pubkey.c' was written
key hash:
7a9db8b522be25688e8d270c543af0846c32faa24eb97d9f00cebbccc75fa74c
```

You need the *key hash* to get your *library license* in the next step.

---

**Tip**

The hash of your public key is included into your ZLM library license to make it very hard to replace the public key in your binary, making your software harder to crack!

---

If you already generated a public/private key pair and want to reuse it, copy the corresponding files `zlm_pubkey.c` and `zlm_privkey.c` into the current directory:

```
cp /path/to/key/files/zlm_pubkey.c .
cp /path/to/key/files/zlm_privkey.c .
```

Afterwards invoke `make` and execute `zlmkeyhash` to get your key hash:

```
make
./zlmkeyhash
```

**Important**

You should create a key pair **only once**, even if you are using multiple platforms or upgrade ZLM! Otherwise older licenses will not work with newer versions of your software!

---

**Get library license** Get your *library license* file `zlm_license.c` from the [ZLM website](#) and copy it into this directory (we assume the file is contained in `~/Downloads/`):

```
cp ~/Downloads/zlm_license.c .
```

---

**Build the ZLM tools and library** Invoke `make` to build the ZLM tools and your ZLM library:

```
make
```

If the `make` invocation was successful the following files have been created.

Table 2: Built ZLM files

libzlm.a	the ZLM library which includes your ZLM library license and public key
api_example	the binary for the API example
zlmhostid	the ZLM hostid tool (you can give this to your customers)
zlmkeyhash	the ZLM keyhash tool which prints the hash of your public key
zlmsign	the ZLM sign tool which signs license files
zlmverify	the ZLM verify tool which verifies the signatures in license files

**Important**

The confidentiality of your private key is very important! Store the `zlm_privkey.c` source file and the `zlm_sign` binary securely!

To test that everything works you can execute the following three commands:

```
zlm_sign example.lic
zlm_verify example.lic
api_example
```

The programs used above are explained in more detail below in section [Section 5](#).

### 3 Installation Windows (Visual Studio)

This section describes the installation for Windows (Visual Studio), see [Section 2](#) for all other platforms (or skip to the [Section 5](#) section).

After you downloaded your ZLM distribution for Windows (Visual Studio) from [ZLM Download](#) you have to extract the archive file (you can use the Windows Explorer for that).

Afterwards open the Visual Studio Command Prompt and change to the freshly created ZLM directory (we assume your username is John, please change accordingly):

```
cd C:\Users\John\Desktop\zlm-1.0-Windows-64bit
```

**Important**

Make sure that the *word size*---32-bit (x86) vs. 64-bit (x64)---of your compiler matches the word size of the used ZLM distribution, otherwise you will get a cryptic message like this: `error LNK2019: unresolved external symbol`

The distribution contains the following files (use `dir` to list the files).

Table 3: The files contained in the distribution

Table 3: (continued)

api_example.c	example program which shows how to use the ZLM API
api_example_dotnet.c	example program which shows how to use the ZLM API from .NET
CHANGELOG	the change log
example.lic	example license file
Makefile	nmake file to build the ZLM tools and the example program
README	a README file containing basic infos
zlm_md.def	module definition file necessary to build DLL (with /MD)
zlm_mt.def	module definition file necessary to build DLL (with /MT)
zlm_md.lib	the ZLM library archive (compiled with /MD)
zlm_mt.lib	the ZLM library archive (compiled with /MT)
zlm.doc.html	a HTML file containing the ZLM documentation
zlm.doc.pdf	the same documentation as a PDF file
zlmgenkeys.exe	the executable to generate the public/private key pair
zlmgenkeys.c	the corresponding source file
zlm.h	the ZLM library header file
zlmhostid.exe	the executable to print/write host IDs
zlmhostid.c	the source file for the zlmhostid tool
zlmkeyhash.c	the source file for the zlmkeyhash tool
zlm_interop_md.cs	C# file containing the .NET interoperability layer (for /MD)
zlm_interop_mt.cs	C# file containing the .NET interoperability layer (for /MT)
zlm_license_dummy.c	needed to build the zlmkeyhash tool
zlm.sign.c	the source file for the zlm.sign tool
zlm.verify.c	the source file for the zlm.verify tools

Now you have to perform three easy steps to set up ZLM:

**Generate public/private key pair** Execute `zlmgenkeys` in this directory to generate a public/private key pair:

```
zlmgenkeys
```

The output of `zlmgenkeys` looks like this:

```
generating keys...
done!
private key file 'zlm_privkey.c' was written
public key file 'zlm_pubkey.c' was written
key hash:
7a9db8b522be25688e8d270c543af0846c32faa24eb97d9f00cebbccc75fa74c
```

You need the *key hash* to get your *library license* in the next step.

#### Tip

The hash of your public key is included into your ZLM library license to make it very hard to replace the public key in your binary, making your software harder to crack!

If you already generated a public/private key pair and want to reuse it, copy the corresponding files `zlm_pubkey.c` and `zlm_privkey.c` into the current directory:

```
copy C:\path\to\key\files\zlm_pubkey.c .
copy C:\path\to\key\files\zlm_privkey.c .
```

Afterwards invoke `nmake` and execute `zlmkeyhash` to get your key hash:

```
nmake
zlmkeyhash
```



### Important

You should create a key pair **only once**, even if you are using multiple platforms or upgrade ZLM! Otherwise older licenses will not work with newer versions of your software!

### Tip

Starting from version 1.8 the Visual Studio distribution of ZLM builds **two** version of all libraries: A /MD version and a /MT version. For example, `libzlm_md.lib` and `libzlm_mt.lib`. Which version you should use depends on the compilation settings of the project you want to use ZLM for. If you are unsure which version to use the /MD version is probably the correct choice. If you are using the wrong version you will get weird linking errors.

**Get library license** Get your *library license* file `zlm_license.c` from the [ZLM website](#) and copy it into this directory (we assume the file is contained in `C:\Users\John\Downloads\`):

```
copy C:\Users\John\Downloads\zlm_license.c .
```

**Build the ZLM tools and library** Invoke `nmake` to build the ZLM tools and your ZLM library:

```
nmake
```

If the `nmake` invocation was successful the following files have been created.

Table 4: Built ZLM files

<code>libzlm_md.lib</code>	the ZLM library which includes your ZLM library license and public key (/MD version)
<code>libzlm_mt.lib</code>	the ZLM library which includes your ZLM library license and public key (/MT version)
<code>api_example_md.exe</code>	the binary for the API example (statically linked with /MD)
<code>api_example_mt.exe</code>	the binary for the API example (statically linked with /MT)
<code>api_example_dll_md.exe</code>	the binary for the API example (linked to /MD version of DLL)
<code>api_example_dll_mt.exe</code>	the binary for the API example (linked to /MT version of DLL)
<code>zlm1_md.dll</code>	the ZLM library as DLL, /MD version (specific your ZLM library license and public key)
<code>zlm1_mt.dll</code>	the ZLM library as DLL, /MT version (includes your ZLM library license and public key)
<code>zlm1_md.exp</code>	the export file which corresponds to the DLL (/MD version)
<code>zlm1_mt.exp</code>	the export file which corresponds to the DLL (/MT version)
<code>zlm1_md.lib</code>	the library file which corresponds to the DLL (/MD version)
<code>zlm1_mt.lib</code>	the library file which corresponds to the DLL (/MT version)
<code>zlmhostid.exe</code>	the ZLM hostid tool (you can give this to your customers)
<code>zlmkeyhash.exe</code>	the ZLM keyhash tool which prints the hash of your public key
<code>zlmshsign.exe</code>	the ZLM sign tool which signs license files
<code>zlmverify.exe</code>	the ZLM verify tool which verifies the signatures in license files



**Important**

The confidentiality of your private key is very important! Store the `zlm_privkey.c` source file and the `zlm_sign.exe` binary securely!

To test that everything works you can execute the following three commands:

```
zlm_sign example.lic
zlm_verify example.lic
api_example_mt
```

The programs used above are explained in more detail below.

**Using the .NET bindings**

The Windows distribution for Visual Studio also contains .NET bindings in the file `zlm_interop_md.cs` (or `zlm_interop_mt.cs`). An example usage of the .NET bindings is shown in `api_example_dotnet.cs`. The .NET bindings use the `zlm_md1.dll` or `zlm_mt1.dll` DLL. If you invoke

```
nmake dotnet
```

A binary `api_example_dotnet_md.exe` is build from `api_example_dotnet.cs` and `zlm_interop_md.cs` and another binary `api_example_dotnet_mt.exe` from `api_example_dotnet.cs` and `zlm_interop_mt.cs`.

**Important**

You should prefer the statically linked `libzlm_md.lib` or `libzlm_mt.lib` over the DLLs `zlm_md1.dll` and `zlm_mt1.dll` wherever possible! The .NET and Java bindings are an exception to this rule.

## 4 ZLM Java bindings

The HTML reference documentation can be found in the `javadoc/` folder of your distribution and [online](#).

Table 5: The files for the ZLM Java bindings contained in the distribution

<code>APIExample.java</code>	example program which shows how to use ZLM Java
<code>APIExample.class</code>	the corresponding class file
<code>zlm1.jar</code>	JAR (Java ARchive) file with the ZLM Java bindings
<code>zlm_jni.a</code>	library file which is used to build the shared library
<code>javadoc/</code>	directory with the Javadoc reference of ZLM Java

On the platforms which contain the ZLM Java bindings a shared library is build during the normal build process (on Windows `zlm1.dll`, on Mac OS `libzlm1.dylib`, and on Linux `libzlm1.so`). To use the ZLM Java bindings you need the `zlm1.jar` file in your classpath and the shared library in your library path. `APIExample.java` shows how to use the ZLM Java bindings in your Java project.

**APIExample.java**

```
import com.zenlicensemanager.zlm.*; /* import all classes from zlm1.jar */

public class APIExample
```

```
{
    public static void main(String[] args)
    {
        ZlmLicense license = null;
        try {
            license = new ZlmLicense();
            license.get("My Product", "1.0", null, ".", null);
        } catch (ZlmException e) {
            System.out.println("error: " + e.getMessage());
            System.exit(1);
        }
        System.out.println("got license!");
        license.free();
    }
    static {
        /* load shared ZLM library */
        if (System.getProperty("os.name").startsWith("Windows")) {
            System.loadLibrary("zlm_mt1");
        } else {
            System.loadLibrary("zlm1");
        }
    }
}
```

If you invoke

```
make java
```

the example program is executed (use `nmake` instead of `make` on Windows).

## 5 ZLM Tools

ZLM contains five command line tools.

**zlmgenkeys** The `zlmgenkeys` tool is usually used only once during the installation of ZLM to generate a public/private key pair (which is saved in the files `zlm_pubkey.c` and `zlm_privkey.c`).

**zlmhostid** The `zlmhostid` tool prints the host ID(s) of the computer it is executed on. The output is in a format which is suitable to be included directly into a license file as a "hostid" value (in order to create a fixed-node license). In addition, it writes its output to the file `zlmhostid.txt` in the current working directory. If the file already exists, it is overwritten. You can freely distribute this tool to your customers.

**zlmkeyhash** The `zlmkeyhash` tool prints the hash of your public key to `stdout`.

**zlmsign** The `zlmsign` tool expects a license file as argument which it signs with your private key. It also automatically includes the appropriate "vendor" field. If no license file is given, `zlmsign` reads the license from `stdin` and writes the result to `stdout`.

If the license file contains a "hostid" entry which is not "any" the host ID(s) are hashed with a random salt and an additional "salt" entry is added to the license file. This makes it harder to spoof MAC addresses.

**zlmverify** The `zlmverify` tool also expects a license file in which it verifies the signatures. It does **not**, however, check any other things about the licenses. That is the "product", "version", "expiry", "hostid", and "customer" fields can contain anything as long as they are in a valid format. Section 6 explains the license format in detail. If no license file is given, `zlmverify` reads the license from `stdin`.

The ZLM tools are just simple wrappers for functions given in the ZLM API and the corresponding source files are included in the ZLM distribution for explanatory purposes, see Section 14.

The ZLM API is described in detail in Section 8.

## 6 License File Format

The license files are written in **JSON** which is easy to read for humans and easy to parse for machines. They contain a single JSON *object* (an unordered set of name/value pairs). The object has two mandatory *members*: the "vendor" and the "license".

### Example license file (single license, unsigned)

```
{
  "vendor": "My Company",
  "license": {
    "product": "My Product",
    "version": "1.0",
    "expiry": "never",
    "hostid": "any",
    "customer": "John Doe"
  }
}
```

**"vendor"** The "vendor" member must have a *string* as value which defines the vendor of the software for which the license is issued. This is set automatically by the `zlm_sign` tool according to your ZLM library license.

**"license"** The "license" member must have either another *object* or an *array* (of *objects*) as value. The license object defines the actual license, in the case of an array it defines multiple licenses, as can be seen in the following example.

### Example license file (multiple licenses, unsigned)

```
{
  "vendor": "ISV",
  "license": [
    {
      "product": "My first Product",
      "version": "2014.12",
      "expiry": "never",
      "hostid":
        "7f7a752390747d36bb6c182b2813cd136cfb19bc31232400328feb9b16b129bb",
      "salt":
        "154a70ff2f22972e33add2aa9def0642ecc19f7eaf5230f21045724a809a46e6",
      "customer": "Big Corp"
    },
    {
      "product": "My second product",
      "version": "2.1",
      "expiry": "2016-01-16",
      "hostid": [
        "67b7dc781748a019064959509cf0743652f6110b6804e3783250743485abb6a6",
        "aacc004aa98cc2a8495822c47689b719aa5172ff58a0c8141550619657f7814b",
        "862e8e6702ff15ccdaca36aa8f7cd89c06cbc7ddda677cc27676546420ffefec"
      ],
      "salt":
        "9c1763e3c7533e1207c385bb5ca946d93d05bca68cd1ac61d0e5eeb1940a0b1b",
      "customer": "Big Corp."
    }
  ]
}
```

The license object itself has multiple mandatory *members* as follows.

**"product"** The "product" member must have a non-empty *string* as value which defines the product of this license. It is compared with the product string supplied to the `zlm_license_get()` method (see Section 8). The product strings are case-sensitive and must match exactly!

**"version"** The "version" member must have a *string* as value which defines the highest-numbered product version supported by this license (in the form "*major.minor*" or "*major.minor.patch*"). The comparison is done numerically: At first the *major*



**"userdata"** A license file can have an optional "userdata" entry which contains a *string* or *object*. Userdata strings or objects are protected against modifications by the license signature, but are not used in ZLM itself. They are useful to extend ZLM, as described below in Section 10.

Please note that the JSON format of the license file requires that some special characters in the userdata string have to be escaped, see [JSON](#) for details. With `zlm_license_userdata()` you get the userdata in escaped form. That is, the caller is responsible for the unescaping. With `zlm_license_userdata_unescaped()` the userdata is returned in unescaped form.

#### Example license file (single license, with userdata)

```
{
  "vendor": "My Company",
  "license": {
    "product": "My Product",
    "version": "1.0",
    "expiry": "never",
    "hostid": "any",
    "customer": "John Doe",
    "userdata": "opaque"
  }
}
```

**"minversion"** A license file can have an optional "minversion" entry. It must have a *string* as value which defines the lowest-numbered product version supported by this license (in the form "*major.minor*" or "*major.minor.patch*"). The comparison is done in the same way as for the "version" entry (however, if the *patch* number is not set it is considered to be 0). "minversion" entries can be used to enforce an update of the software to a minimal version number.

#### Example license file (single license, with minversion)

```
{
  "vendor": "My Company",
  "license": {
    "product": "My Product",
    "version": "1.1",
    "expiry": "never",
    "hostid": "any",
    "customer": "John Doe",
    "minversion": "1.1"
  }
}
```

**"os"** A license file can have an optional "os" entry. It must have a *string* as value which defines the allowed operating system for this license (either "Android", "FreeBSD", "iOS", "Linux", "Mac OS", or "Windows"). Thereby, "os" entries can be used to bind a license to a certain operating system.

#### Example license file (single license, with os)

```
{
  "vendor": "Wikena GmbH",
  "license": {
    "product": "My Product",
    "version": "1.1",
    "expiry": "never",
    "hostid": "any",
    "os": "Linux",
    "customer": "John Doe"
  }
}
```

**"cpu"** A license file can have an optional "cpu" entry. It must have a *string* as value which defines the allowed CPU architecture for this license (either "x86\_64", "x86", "aarch64", "arm", "powerpc64", "powerpc64le", "powerpc32", "powerpc32le", "mips64", "mips64el", "+mips32", or "mips32el"). Thereby, "cpu" entries can be used to bind a license to a certain CPU architecture.

**Example license file (single license, with cpu)**

```
{
  "vendor": "Wikena GmbH",
  "license": {
    "product": "My Product",
    "version": "1.1",
    "expiry": "never",
    "hostid": "any",
    "cpu": "x86_64",
    "customer": "John Doe"
  }
}
```

**"appcert" (Android only)** The "appcert" member must have an *array* (of *strings*) as value which denote all the application signing certificate SHA-256 hashes for this license file. The license will work only with applications which have been signed by one of the given signing certificates.

**Example license file (license, with appcert)**

```
{
  "vendor": "Wikena GmbH",
  "license": {
    "product": "My Product",
    "version": "1.1",
    "expiry": "never",
    "hostid": "any",
    "customer": "John Doe",
    "appcert": [
      "E1:6C:72:6F:0B:07:96:3C:A0:5E:7A:32:46:5F:69:ED:CC:63:CC:5D:F4:D0:E2:83:C7:EC:15:54: ←
        BF:94:ED:48",
      "F2:43:AA:14:0E:D2:59:38:C7:4D:8C:31:FC:2E:48:85:48:DC:5C:94:DE:4E:45:F1:BA:64:0C:BA: ←
        D9:94:95:74"
    ]
  }
}
```

See Section 12 for further information.

## 7 Integrating ZLM

The file `api_example.c` shows how easy it is to integrate ZLM into your software product.

**api\_example.c**

```
#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

int main(int argc, char *argv[])
{
    ZlmLicense *license;
    char err[ZLM_ERRBUF];

    /* create license object */
    if (!(license = zlm_license_new(err))) {
        fprintf(stderr, "error: %s\n", err);
        return EXIT_FAILURE;
    }

    /* try to get license for product 'My Product' from the same directory as the
```

```
    executed binary and if that fails from the current directory */
if (zlm_license_get(license, "My Product", "1.0", argv[0], ".", NULL, err)) {
    fprintf(stderr, "error: %s\n", err);
    zlm_license_free(license);
    return EXIT_FAILURE;
}
else
    puts("got license!");

/*****
 * run your code here *
*****/

/* free license object */
zlm_license_free(license);

return EXIT_SUCCESS;
}
```

Integrating ZLM consists of three simple steps:

1. In the beginning of your program you call `zlm_license_new()` to create a new `ZlmLicense` object.
2. Afterwards you call `zlm_license_get()` to retrieve a license.
3. In the end of your program you call `zlm_license_free()` to free the license object.

When building your binary, just link against the ZLM library file `libzlm.a`, which does not have any other dependencies, and you are ready to go!

Please refer to next section for more details on the ZLM API.

## 8 API Documentation

The `zlm.h` header file contains everything you need to use the ZLM API

```
/*
 Copyright (c) 2013-2020 Wiken GmbH <support@zenlicensemanager.com>
 All rights reserved.
 */

#ifndef ZLM_H
#define ZLM_H

#ifdef __cplusplus
extern "C" {
#endif

#ifdef __ANDROID__
#include <jni.h>
#endif

#ifdef __APPLE__
#include <TargetConditionals.h>
#endif

/* The minimum size of an error buffer. */
#define ZLM_ERRBUF 1024

/* The minimum size of a salt string. */
```

```

#define ZLM_SALTLEN 65

/* ZLM error codes. */
#define ZLM_OK 0 /* everything went fine */
#define ZLM_NOMEM -2 /* memory allocation error */
#define ZLM_IOERR -3 /* I/O error */
#define ZLM_APIERR -4 /* API was used incorrectly */
#define ZLM_NOLIBLIC -5 /* library license malformed/invalid */
#define ZLM_NOLICFILE -6 /* no license file found */
#define ZLM_BADLICFILE -7 /* malformed licfile */
#define ZLM_BADVENDOR -8 /* vendor mismatch between library lic. and file */
#define ZLM_NOVERIFY -9 /* signature verification failed */
#define ZLM_OLDVERSION -10 /* version in license is too old */
#define ZLM_EXPIRED -11 /* the license expired */
#define ZLM_WRONGHOST -12 /* wrong host ID / appcert */
#define ZLM_NOPRODUCT -13 /* no license found for product */
#define ZLM_HOSTIDERR -14 /* error while determining host ID */
#define ZLM_KEYGENERR -15 /* error while generating public/private key pair */
#define ZLM_SIGNERR -16 /* error while signing */
#define ZLM_MINVERSION -17 /* product version is too old for minversion */
#define ZLM_WRONGOS -18 /* wrong operating system */
#define ZLM_JNIERR -19 /* JNI related error (Android only) */
#define ZLM_WRONGCPU -20 /* wrong CPU architecture */

/* The type of a ZLM license object. */
typedef struct ZlmLicense ZlmLicense;

/* Return a new license object (or NULL if not enough memory is available). */
ZlmLicense* zlm_license_new(char *err);

/* Try to get a new <license> for the given <product>.
The version given in the corresponding license file or string must be smaller
or equal than the one given in the mandatory <product_version> parameter such
that the license is valid. <product_version> must be of the form
"major.minor" (e.g., "1.0"). If no valid license can be obtained an error
code is returned and an error message describing the problem is stored in
<err> (if <err> is not NULL). If <err> is not NULL, it must have at least
ZLM_ERRBUF many bytes. If a valid license was obtained, ZLM_OK is returned
and <license> points to the active license.
zlm_license_get() tries to acquire the license in the following order:

1.) If the environment variable <ZLM_LICENSE> is defined it is used as path:
If it points directly to a file, the file is probed for a valid license.
If it points to a directory, then all files ending with '.lic' are probed
for a valid license.

2.) If <argv0> is not NULL it should point to the first member of your
main's argv. In this case, the directory in which the main binary resides
is used as license directory.

3.) If <path> is not NULL it is used as license path (as in case 1.)).
Use "." to search the current directory for valid license files.

4.) If <license_string> is not NULL the given string is used as the content
of a license file.

zlm_license_get() can only be called once on a fresh <license> object. */
int zlm_license_get(ZlmLicense *license, const char *product,
                  const char *product_version, const char *argv0,
                  const char *path, const char *license_string,
                  char *err);

/* Release <license> and free corresponding memory. */
void zlm_license_free(ZlmLicense *license);

```



```
/* Return the product of the given <license>. <license> must be a license which
   has been validated with zlm_license_get(), otherwise NULL is returned. */
const char* zlm_license_product(const ZlmLicense *license);

/* Return the version of the given <license>. <license> must be a license which
   has been validated with zlm_license_get(), otherwise NULL is returned. */
const char* zlm_license_version(const ZlmLicense *license);

/* Return the expiry of the given <license>. <license> must be a license which
   has been validated with zlm_license_get(), otherwise NULL is returned. */
const char* zlm_license_expiry(const ZlmLicense *license);

/* Return the number of days the given <license> will expire in. Returns 0 if
   the license never expires. <license> must be a license which has been
   validated with zlm_license_get(), otherwise ZLM_APIERR is returned. */
int zlm_license_expiry_days(const ZlmLicense *license);

/* Return the customer of the given <license>. <license> must be a license which
   has been validated with zlm_license_get(), otherwise NULL is returned. */
const char* zlm_license_customer(const ZlmLicense *license);

/* Return the userdata associated with the given valid <license>. Possible JSON
   escape sequences are included in unmodified form. If the <license> has not
   been validated with zlm_license_get() or it does not have userdata
   associated with it NULL is returned. */
const char* zlm_license_userdata(const ZlmLicense *license);

/* Return the userdata associated with the given valid <license> in unescaped
   form. That is, possible JSON escape sequences have been removed: The escaped
   characters '\"', '\\', '\/', '\b', '\f', '\n', '\r', and '\t' have been
   unescaped and '\uXXXX' sequences have been converted to 1-3 bytes UTF-8.
   If the <license> has not been validated with zlm_license_get(), it does not
   have userdata associated with it, or no memory could be allocated NULL is
   returned. The caller is responsible to free the returned string with
   zlm_free()! */
char* zlm_license_userdata_unescaped(const ZlmLicense *license);

/* Sets <license> to the next available valid license.
   <license> must be a license which has been validated with
   zlm_license_get(), otherwise ZLM_APIERR is returned.
   If no further valid license can be obtained an error code is returned and an
   error message describing the problem is stored in <err> (if <err> is not
   NULL). If <err> is not NULL, it must have at least ZLM_ERRBUF many bytes.
   This method can be used to extend the ZLM checks with your own userdata. For
   example, if you want to implement different host IDs.*/
int zlm_license_next(ZlmLicense *license, char *err);

/* Check if <license> has expired. <license> must be a license which has been
   validated with zlm_license_get(), otherwise ZLM_APIERR is returned. If the
   license expired since it was validated with zlm_license_get(), ZLM_EXPIRED
   is returned and an error message describing the expiry is stored in <err> (if
   <err> is not NULL). If <err> is not NULL, it must have at least ZLM_ERRBUF
   many bytes. */
int zlm_license_not_expired(const ZlmLicense *license, char *err);

/* Return the version of the ZLM library in use as a string. */
const char* zlm_version(void);

/* Return the unhashed host ID string of this computer in JSON format (in a
   single line). The caller is responsible to free the returned string with
   zlm_free()! In case of error, NULL is returned and the error buffer <err> is
   set accordingly. */
```

```
char*      zlm_hostid_json(char *err);

/* Return the hashed host ID string of this computer in JSON format (in a
   single line). The caller is responsible to free the returned string with
   zlm_free()! In case of error, NULL is returned and the error buffer <err> is
   set accordingly. The salt used for the hashing is stored in <salt> which must
   not be NULL and have at least ZLM_SALTLEN many bytes.
   Deprecated: Use zlm_hostid_json() instead! */
char*      zlm_hostid_json_hashed(char *salt, char *err);

/* Free the memory space pointed to by <ptr>, which must have been
   returned by a previous call to another ZLM library function. */
void       zlm_free(void *ptr);

/* License check methods which make it harder to remove ZLM from your binary.
   You should call these methods in different places of your program with a
   valid <license>. Thereby, <license> must be a license which has been
   validated with zlm_license_get(), otherwise the methods will crash!
   zlm_license_check_a() to zlm_license_check_f() just check, if the license was
   valid before.
   zlm_license_check_g() to zlm_license_check_l() also recheck the expiry.
   That is, if the license expired in the meantime, these checks would crash.
   If you want to shut down gracefully in such cases, use
   zlm_license_not_expired() instead to recheck the expiry.
   zlm_license_check_m() to zlm_license_check_r() also recheck the hostid
   (but not the expiry). That is, if the hostid became invalid in the meantime,
   these checks would crash. */
void       zlm_license_check_a(const ZlmLicense *license);
void       zlm_license_check_b(const ZlmLicense *license);
void       zlm_license_check_c(const ZlmLicense *license);
void       zlm_license_check_d(const ZlmLicense *license);
void       zlm_license_check_e(const ZlmLicense *license);
void       zlm_license_check_f(const ZlmLicense *license);
void       zlm_license_check_g(const ZlmLicense *license);
void       zlm_license_check_h(const ZlmLicense *license);
void       zlm_license_check_i(const ZlmLicense *license);
void       zlm_license_check_j(const ZlmLicense *license);
void       zlm_license_check_k(const ZlmLicense *license);
void       zlm_license_check_l(const ZlmLicense *license);
void       zlm_license_check_m(const ZlmLicense *license);
void       zlm_license_check_n(const ZlmLicense *license);
void       zlm_license_check_o(const ZlmLicense *license);
void       zlm_license_check_p(const ZlmLicense *license);
void       zlm_license_check_q(const ZlmLicense *license);
void       zlm_license_check_r(const ZlmLicense *license);

/* The type of a key function (public or private). */
typedef const unsigned char* (ZlmKeyfunc)(int *keylen);

/* Generate public/private key pair and write it to the files zlm_pubkey.c and
   zlm_privkey.c in the current directory. */
int zlm_genkeys(char *err);

/* Print host ID string of this computer on stdout. */
int zlm_hostid(char *err);

/* Print host ID string of this computer on stdout and to write it to a file
   named zlmhostid.txt in the current working directory.
   If the file already exists it is overwritten! */
int zlm_hostid_file(char *err);

/* Print hash of <public_key> on stdout. */
```

```
int zlm_keyhash(ZlmKeyfunc *public_key, char *err);

/* Sign the license file given in <path> with the given <private_key>.
   If <path> is NULL the license file is read from stdin and the result written
   to stdout. */
int zlm_sign_file(const char *path, ZlmKeyfunc *private_key, char *err);

/* Verify signatures of license file given in <path> with given <public_key>.
   If <path> is NULL the license file is read from stdin. */
int zlm_verify_file(const char *path, ZlmKeyfunc *public_key, char *err);

/* Sign the license file given in buffer <license> with the given <private_key>
   and return the result. The caller is responsible to free the returned string
   with zlm_free()! In case of error, NULL is returned and the error buffer
   <err> is set accordingly. */
char* zlm_sign_license(const char *license, ZlmKeyfunc *private_key, char *err);

#ifdef __ANDROID__
/* Android only: In order to use ANDROID_ID entries in the "hostid" field of
   license files you have to call this this function once before using any other
   library function! <env> must be the JNI environment, otherwise ZLM_APIERR is
   returned. <activity> must be an Android activity (that is, an object with the
   getContentResolver() method), otherwise ZLM_APIERR or ZLM_JNIERR is returned.
   If the ANDROID_ID cannot be determined ZLM_JNIERR is returned and err
   contains an error message. */
int zlm_read_android_id(JNIEnv *env, jobject activity, char *err);

/* Android only: Return the ANDROID_ID. Before this method can be used
   the ANDROID_ID has to be read once by calling zlm_read_android_id(). */
const char* zlm_android_id(void);

/* Android only: In order to use application ID entries in the "hostid" field of
   license files you have to call this this function once before using any other
   library function! <env> must be the JNI environment, otherwise ZLM_APIERR is
   returned. <activity> must be an Android activity (that is, an object with the
   getPackageName() method), otherwise ZLM_APIERR or ZLM_JNIERR is returned.
   If the application ID cannot be determined ZLM_JNIERR is returned and err
   contains an error message. */
int zlm_read_application_id(JNIEnv *env, jobject activity, char *err);

/* Android only: Return the application ID. Before this method can be used the
   application ID has to be read once by calling zlm_read_application_id(). */
const char* zlm_application_id(void);

/* Android only: In order to use application certificate entries in the
   "appcert" field of license files you have to call this this function once
   before using any other library function! <env> must be the JNI environment,
   otherwise ZLM_APIERR is returned. If the application certificate cannot be
   determined ZLM_JNIERR is returned and err contains an error message. */
int zlm_read_application_cert(JNIEnv *env, jobject activity, char *err);

/* Android only: Return the (oldest) application certificate as SHA-256.
   Before this method can be used the application ID has to be read once by
   calling zlm_read_application_cert(). */
const char* zlm_application_cert(void);
#endif

#if defined (__APPLE__) && (TARGET_OS_IOS > 0)
/* iOS only: Return the vendor specific unique key (a UUID conforming to RFC
   4122 version 4) as a C-string. See http://apple.co/2ohHcQY for details.

   If the result is NULL wait and get the value again later. This happens, for
```

example, after the device has been restarted but before the user has unlocked the device.

The value in this property remains the same while the app (or another app from the same vendor) is installed on the iOS device. The value changes when the user deletes all of that vendor's apps from the device and subsequently reinstalls one or more of them. The value can also change when installing test builds using Xcode or when installing an app on a device using ad-hoc distribution. \*/

```
const char* zlm_ios_hostid(char *err);

/* iOS only: Return the unique bundle identifier (CFBundleIdentifier).
   See https://apple.co/2SU5j50 for details. */
const char* zlm_ios_bundle_identifier(char *err);
#endif

#ifdef __cplusplus
}
#endif

#endif
```

The ZLM API consists of the following parts.

**Error buffer** ZLM uses a `char*` as an error buffer to store error messages. If you want to get error messages you have to define a buffer with a minimum size of `ZLM_ERRBUF`:

```
char err[ZLM_ERRBUF];
```

and pass `err` to the ZLM functions which take an error object. If you are not interested in the actual error message you can just pass `NULL` and use the error codes.

**Error codes** All functions which take an `err` buffer return a negative `int` to indicate an error. If everything went fine `ZLM_OK` is returned.

**The license object** The `ZlmLicense` type defines the major class in ZLM. It is used to represent a license. At first a license object has to be created with `zlm_license_new()`, which returns a new (unvalidated) license object (or `NULL` if not enough memory is available):

```
ZlmLicense* zlm_license_new(char *err);
```

For example like this:

```
ZlmLicense *license;
char err[ZLM_ERRBUF];

if (!(license = zlm_license_new(err))) {
    fprintf(stderr, "error: %s\n", err);
    return EXIT_FAILURE;
}
```

`zlm_license_new()` **only** returns `NULL` if not enough memory was available to create the object.

Afterwards you try to get a valid license with `zlm_license_get(license)` for a given product. The version given in the corresponding license file or string must be smaller or equal than the one given in the mandatory version parameter such that the license is valid. The version must be of the form "*major.minor*" (e.g., "1.0"). If no valid license can be obtained, an error code is returned and an error message describing the problem is stored in `err` (if `err` is not `NULL`). `zlm_license_get()` tries to acquire the license in the following order:

1. If the environment variable `ZLM_LICENSE` is defined it is used as path: If it points directly to a file, the file is probed for a valid license. If it points to a directory, then all files ending with `.lic` are probed for a valid license.

2. If `argv0` is not `NULL` it should point to the first member of your main's `argv`. In this case, the directory in which the main binary resides is used as license directory.
3. If `path` is not `NULL` it is used as license path (as in case 1.)). Use `"."` to search the current directory for valid license files.
4. If `license_string` is not `NULL` the given string is used as the content of a license file.

```
int zlm_license_get(ZlmLicense *license,
                   const char *product,
                   const char *version,
                   const char *argv0,
                   const char *path,
                   const char *license_string,
                   char *err);
```

For example to get a license for version "1.0" of the product "My Product" from the same directory as the executed binary and if that fails from the current directory, you call `zlm_license_get()` like this:

```
if (zlm_license_get(license, "My Product", "1.0", argv[0], ".", NULL, err)) {
    fprintf(stderr, "error: %s\n", err);
    zlm_license_free(license);
    return EXIT_FAILURE;
}
else
    puts("got license!");
```

The call above is the most common use of `zlm_license_get()`.

Finally, you have to call `zlm_license_free()` at the end of your program to release the license and free the corresponding memory:

```
zlm_license_free(license);
```

**Methods on the license object** If you have a valid license object, you can get information about the license with various helper methods:

Return the product of the given license. `license` must be a license which has been validated with `zlm_license_get()`, otherwise `NULL` is returned.

```
const char* zlm_license_product(const ZlmLicense *license);
```

Return the version of the given license. `license` must be a license which has been validated with `zlm_license_get()`, otherwise `NULL` is returned.

```
const char* zlm_license_version(const ZlmLicense *license);
```

Return the expiry of the given license. `license` must be a license which has been validated with `zlm_license_get()`, otherwise `NULL` is returned.

```
const char* zlm_license_expiry(const ZlmLicense *license);
```

Return the number of days the given license will expire in. Returns 0 if the license never expires. `license` must be a license which has been validated with `zlm_license_get()`, otherwise `ZLM_APIERR` is returned.

```
int zlm_license_expiry_days(const ZlmLicense *license);
```

Return the customer of the given license. `license` must be a license which has been validated with `zlm_license_get()`, otherwise `NULL` is returned.

```
const char* zlm_license_customer(const ZlmLicense *license);
```

**Extension methods** The following two methods are only useful if you want to extend ZLM, as described below in Section 10.

Return the userdata associated with the given valid `license`. If the `license` has not been validated with `zlm_license_get()` or it does not have userdata associated with it `NULL` is returned.

```
const char *zlm_license_userdata(const ZlmLicense *license);
```

Return the userdata associated with the given valid `license` in unescaped form. That is, possible JSON escape sequences have been removed: The escaped characters `\", \\, \b, \f, \n, \r, and \t` have been unescaped and `\uXXXX` sequences have been converted to 1-3 bytes UTF-8. If the `license` has not been validated with `zlm_license_get()`, it does not have userdata associated with it, or no memory could be allocated `NULL` is returned. The caller is responsible to free the returned string with `zlm_free()`!

```
char* zlm_license_userdata_unescaped(const ZlmLicense *license);
```

Sets `license` to the next available valid license. `license` must be a license which has been validated with `zlm_license_get()`, otherwise `ZLM_APIERR` is returned. If no further valid license can be obtained, an error code is returned and an error message describing the problem is stored in `err` (if `err` is not `NULL`). If `err` is not `NULL`, it must have at least `ZLM_ERRBUF` many bytes. This method can be used to extend the ZLM checks with your own userdata. For example, if you want to implement different host IDs.

```
int zlm_license_next(ZlmLicense *license, char *err);
```

**Version function** Return the version of the ZLM library in use as a string.

```
const char* zlm_version(void);
```

**Hostid (JSON) function** Return the host ID string of this computer in JSON format (in a single line). The caller is responsible to free the returned string with `zlm_free()`! In case of error, `NULL` is returned and the error buffer `err` is set accordingly.

```
char* zlm_hostid_json(char *err);
```

This function can be helpful in automatic license generation for a certain host computer.

**Hostid (JSON) function (hashed)** Return the hashed host ID string of this computer in JSON format (in a single line). The caller is responsible to free the returned string with `zlm_free()`! In case of error, `NULL` is returned and the error buffer `err` is set accordingly. The salt used for the hashing is stored in `salt` which must not be `NULL` and have at least `ZLM_SALTLEN` many bytes.

```
char* zlm_hostid_json_hashed(char *salt, char *err);
```



### Important

`zlm_hostid_json_hashed()` has been deprecated and `zlm_hostid_json()` should be used instead!

---

**Free function** Free the memory space pointed to by `ptr`, which must have been returned by a previous call to another ZLM library function.

```
void zlm_free(void *ptr);
```

Using `zlm_free()` instead of the `stdlib free()` helps to prevent problems for some Windows configurations (and doesn't hurt in other situations).

**License check methods** The license check methods make it harder to remove ZLM from your binary. You should call these methods in different places of your program with a valid `license`. Thereby, `license` must be a license which has been validated with `zlm_license_get()`, otherwise the methods will crash!

---

```

void zlm_license_check_a(const ZlmLicense *license);
void zlm_license_check_b(const ZlmLicense *license);
void zlm_license_check_c(const ZlmLicense *license);
void zlm_license_check_d(const ZlmLicense *license);
void zlm_license_check_e(const ZlmLicense *license);
void zlm_license_check_f(const ZlmLicense *license);
void zlm_license_check_g(const ZlmLicense *license);
void zlm_license_check_h(const ZlmLicense *license);
void zlm_license_check_i(const ZlmLicense *license);
void zlm_license_check_j(const ZlmLicense *license);
void zlm_license_check_k(const ZlmLicense *license);
void zlm_license_check_l(const ZlmLicense *license);
void zlm_license_check_m(const ZlmLicense *license);
void zlm_license_check_n(const ZlmLicense *license);
void zlm_license_check_o(const ZlmLicense *license);
void zlm_license_check_p(const ZlmLicense *license);
void zlm_license_check_q(const ZlmLicense *license);
void zlm_license_check_r(const ZlmLicense *license);

```

**Tip**

You should spread calls of this methods (use different ones) throughout your code base to make the removal of ZLM from your binary harder. See Section 9 for example code. If somebody removes the `zlm_license_get()` call from your binary, this method calls will crash the binary.

**Key function** The type `ZlmKeyfunc` is used to pass public and private keys to the functions who need them.

**Tool functions** The tool functions give you the functionality of the ZLM tools in the API itself. They all take an `err` buffer and return an error code (described above):

Generate public/private key pair and write it to the files `zlm_pubkey.c` and `zlm_privkey.c` in the current directory.

```
int zlm_genkeys(char *err);
```

Print host ID string of this computer on stdout.

```
int zlm_hostid(char *err);
```

Print host ID string of this computer on stdout and to write it to a file named `zlmhostid.txt` in the current working directory. If the file already exists it is overwritten!

```
int zlm_hostid_file(char *err);
```

Print hash of `public_key` on stdout.

```
int zlm_keyhash(ZlmKeyfunc *public_key, char *err);
```

Sign the license file given in `path` with the given `private_key`.

```
int zlm_sign_file(const char *path, ZlmKeyfunc *private_key, char *err);
```

Verify signatures of license file given in `path` with given `public_key`.

```
int zlm_verify_file(const char *path, ZlmKeyfunc *public_key, char *err);
```

The code of the actual ZLM tools which use the tool APIs is given in Section 5.

**Sign in memory function**

Sign the license file given in buffer `license` with the given `private_key` and return the result. The caller is responsible to free the returned string with `zlm_free()`! In case of error, NULL is returned and the error buffer `err` is set accordingly.

```
char* zlm_sign_license(const char *license, ZlmKeyfunc *private_key, char *err);
```

To use this function in the backend one would define:

```
extern const unsigned char* zlm_get_private_key(int *keylen);
```

and link against the object file `zlm_privkey.o` (compiled from `zlm_privkey.c`) and then pass `zlm_get_private_key` to `zlm_sign_license()` as the `private_key` argument.



### Important

**Never** link `zlm_privkey.o` into binaries distributed to customers!

## 9 More API Examples

This section contains more example programs which use the major ZLM API `zlm_get_license()` in different ways and explain the usage of the `zlm_license_check_x()` methods. The following program tries to get the license from a user supplied license file path.

### api\_example\_path.c

```
#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

int main(int argc, char *argv[])
{
    ZlmLicense *license;
    char err[ZLM_ERRBUF];

    if (argc != 2) {
        fprintf(stderr, "Usage %s license_file\n", argv[0]);
        return EXIT_FAILURE;
    }

    /* create license object */
    if (!(license = zlm_license_new(err))) {
        fprintf(stderr, "error: %s\n", err);
        return EXIT_FAILURE;
    }

    /* try to get license for product 'My Product' from supplied license file path
       (first argument in argument vector) */
    if (zlm_license_get(license, "My Product", "1.0", NULL, argv[1], NULL, err)) {
        fprintf(stderr, "error: %s\n", err);
        zlm_license_free(license);
        return EXIT_FAILURE;
    }
    else
        puts("got license!");

    /* *****
     * run your code here *
     * ***** */

    /* free license object */
    zlm_license_free(license);
```



```

    return EXIT_SUCCESS;
}

```

And in the following example the license is passed directly to `zlm_license_get()` as a license string. This technique can be used to protect a software library with ZLM.

#### api\_example\_string.c

```

#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

#define LICENSE \
    "{ \"vendor\": \"Wikena GmbH\", \"license\": { \"product\": \"My Product\", \" \
    \"version\": \"1.0\", \"expiry\": \"never\", \"hostid\": \"any\", \" \
    \"customer\": \"John Doe\", \"signature\": \"3041021e74901fc7088ae32bb1\" \
    \"276dee2699d6d4e82cd5b39df396a57f608ffa31be021f008d59f1d76a0ac31ad85f\" \
    \"9428f145df50292748f15834f92c2d557087b1af\" } }"

int main(int argc, char *argv[])
{
    ZlmLicense *license;
    char err[ZLM_ERRBUF];

    /* create license object */
    if (!(license = zlm_license_new(err))) {
        fprintf(stderr, "error: %s\n", err);
        return EXIT_FAILURE;
    }

    /* try to get license for product 'My Product' directly from supplied license
       string */
    if (zlm_license_get(license, "My Product", "1.0", NULL, NULL, LICENSE, err)) {
        fprintf(stderr, "error: %s\n", err);
        zlm_license_free(license);
        return EXIT_FAILURE;
    }
    else
        puts("got license!");

    /* *****
       * run your code here *
       ***** */

    /* free license object */
    zlm_license_free(license);

    return EXIT_SUCCESS;
}

```

In the following example a license is acquired and checked multiple times throughout the code. The `license` object is passed around explicitly.

#### api\_example\_check.c

```

#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

static void foo(ZlmLicense *license)
{
    zlm_license_check_a(license);
}

```

```
    /*****
    * run your code here *
    *****/
}

static void bar(ZlmLicense *license)
{
    zlm_license_check_b(license);

    /*****
    * run your code here *
    *****/
}

static void baz(ZlmLicense *license)
{
    zlm_license_check_c(license);

    /*****
    * run your code here *
    *****/
}

int main(int argc, char *argv[])
{
    ZlmLicense *license;
    char err[ZLM_ERRBUF];

    /* create license object */
    if (!(license = zlm_license_new(err))) {
        fprintf(stderr, "error: %s\n", err);
        return EXIT_FAILURE;
    }

    /* try to get license for product 'My Product' from the same directory as the
       executed binary and if that fails from the current directory */
    if (zlm_license_get(license, "My Product", "1.0", argv[0], ".", NULL, err)) {
        fprintf(stderr, "error: %s\n", err);
        zlm_license_free(license);
        return EXIT_FAILURE;
    }
    else
        puts("got license!");

    /*****
    * example code foo(), bar(), baz() called here *
    *****/
    foo(license);
    bar(license);
    baz(license);

    /* free license object */
    zlm_license_free(license);

    return EXIT_SUCCESS;
}
```

The following example is similar to the previous one, but the `license` object is stored in a *global* variable.

#### **api\_example\_global.c**

```
#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

ZlmLicense *license; /* global license variable */

static void foo(void)
{
    zlm_license_check_a(license);

    /******
    * run your code here *
    *****/
}

static void bar(void)
{
    zlm_license_check_b(license);

    /******
    * run your code here *
    *****/
}

static void baz(void)
{
    zlm_license_check_c(license);

    /******
    * run your code here *
    *****/
}

int main(int argc, char *argv[])
{
    char err[ZLM_ERRBUF];

    /* create global license object */
    if (!(license = zlm_license_new(err))) {
        fprintf(stderr, "error: %s\n", err);
        return EXIT_FAILURE;
    }

    /* try to get license for product 'My Product' from the same directory as the
       executed binary and if that fails from the current directory */
    if (zlm_license_get(license, "My Product", "1.0", argv[0], ".", NULL, err)) {
        fprintf(stderr, "error: %s\n", err);
        zlm_license_free(license);
        return EXIT_FAILURE;
    }
    else
        puts("got license!");

    /******
    * example code foo(), bar(), baz() called here *
    *****/
    foo();
    bar();
    baz();

    /* free license object */
    zlm_license_free(license);
}
```

```
    return EXIT_SUCCESS;
}
```

## 10 Extending ZLM

The methods `zlm_license_userdata()` and `zlm_license_next()` (see Section 8) can be used to *extend* ZLM. The following example shows how to extend ZLM with a simple hostname check. That is, the hostname stored in "userdata" must match the hostname of the system the program is executed on.

We use `zlm_license_userdata()` to get the hostname from the "userdata" field and `zlm_license_next()` to iterate through all valid licenses (ZLM itself does not consider the "userdata" field) until we find one where the hostname matches.

### api\_example\_userdata.c

```
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifdef _WIN32
#include <unistd.h>
#else
#include <winsock2.h>
#endif
#include "zlm.h"

int main(int argc, char *argv[])
{
    ZlmLicense *license;
    char hostname[256], err[ZLM_ERRBUF];
    int had_err = ZLM_OK;

    /* determine hostname */
    if (gethostname(hostname, sizeof hostname)) {
        fprintf(stderr, "error: cannot determine hostname: %s\n", strerror(errno));
        return EXIT_FAILURE;
    }

    /* create license object */
    if (!(license = zlm_license_new(err))) {
        fprintf(stderr, "error: %s\n", err);
        return EXIT_FAILURE;
    }

    /* try to get license for product 'My Product' from the same directory as the
       executed binary and if that fails from the current directory */
    if (zlm_license_get(license, "My Product", "1.0", argv[0], ".", NULL, err)) {
        fprintf(stderr, "error: %s\n", err);
        zlm_license_free(license);
        return EXIT_FAILURE;
    }

    /* we got the first valid license */
    do {
        /* get userdata */
        const char *userdata = zlm_license_userdata(license);
        if (userdata) {
            /* check if the userdata contains the hostname of this machine */
            if (strcmp(userdata, hostname) == 0) {
```

```
        /* valid license found, break while-loop */
        break;
    }
}
/* try to get next valid license */
had_err = zlm_license_next(license, err);
} while (had_err == ZLM_OK);

if (had_err) {
    fprintf(stderr, "error: %s\n", err);
    zlm_license_free(license);
    return EXIT_FAILURE;
}
else
    puts("got license!");

/*****
 * run your code here *
*****/

/* free license object */
zlm_license_free(license);

return EXIT_SUCCESS;
}
```

## 11 Setting up MinGW

The installation for Windows (MinGW) is described in Section 2, this section describes how to setup your MinGW environment on Windows.

1. Download the MSYS2 installer from <http://www.msys2.org/> (usually you want the x86\_64 installer no matter what you want to compile, unless you are still running a 32bit version of Windows).
2. Run the installer and install MSYS2 into C:\msys64 (or C:\msys32 for the 32bit version) and launch the MSYS2 shell (C:\msys64\msys2).
3. In the MSYS2 shell enter: `pacman -Sy pacman` (to update the package DB and pacman)
4. Close the MSYS2 shell and launch it again (C:\msys64\msys2).  
In the MSYS2 shell enter: `pacman -Syu` (to update the package DB and install the core system packages).
5. Close the MSYS2 shell and launch it again (C:\msys64\msys2).  
Update the rest with `pacman -Su`
6. Close the MSYS2 shell and launch it again (C:\msys64\msys2).  
Install MinGW 64bit: `pacman -S mingw-w64-x86_64-gcc` (to compile ZLM 64bit).  
Install MinGW 32bit: `pacman -S mingw-w64-i686-gcc` (to compile ZLM 32bit).
7. Install make: `pacman -S make`.
8. Optionally install any libraries/tools you may need.  
You can search the repository by `pacman -Ss name_of_package`  
and install a package by `pacman -S name_of_package`.
9. To compile the ZLM Windows (MinGW) 64bit launch C:\msys64\mingw64  
and follow guidelines in Section 2.  
For 32bit launch C:\msys64\mingw32

## 12 Android Specifics

To build `libzlm.a` you have to set some environment variables pointing to your Android NDK. Please refer to the `Makefile` in your Android distribution for details.

On Android it is also possible to use an `ANDROID_ID` or application ID as `"hostid"` entry in license files (see Section 6). Application IDs must be prefixed with `"applicationID="`.

**Android only APIs** In order to use `ANDROID_ID` entries in the `"hostid"` field of license files you have to call this function once before using any other library function! `env` must be the JNI environment, otherwise `ZLM_APIERR` is returned. `activity` must be an Android activity (that is, an object with the `getContentResolver()` method), otherwise `ZLM_APIERR` or `ZLM_JNIERR` is returned. If the `ANDROID_ID` cannot be determined `ZLM_JNIERR` is returned and `err` contains an error message.

```
int zlm_read_android_id(JNIEnv *env, jobject activity, char *err);
```

Return the `ANDROID_ID`. Before this method can be used the `ANDROID_ID` has to be read once by calling `zlm_read_android_id()`.

```
const char* zlm_android_id(void);
```

In order to use application ID entries in the `"hostid"` field of license files you have to call this function once before using any other library function! `env` must be the JNI environment, otherwise `ZLM_APIERR` is returned. `activity` must be an Android activity (that is, an object with the `getPackageName()` method), otherwise `ZLM_APIERR` or `ZLM_JNIERR` is returned. If the application ID cannot be determined `ZLM_JNIERR` is returned and `err` contains an error message.

```
int zlm_read_application_id(JNIEnv *env, jobject activity, char *err);
```

Return the application ID. Before this method can be used the application ID has to be read once by calling `zlm_read_application_id()`.

```
const char* zlm_application_id(void);
```

In order to use application certificate entries in the `"appcert"` field of license files you have to call this function once before using any other library function! `env` must be the JNI environment, otherwise `ZLM_APIERR` is returned. If the application certificate cannot be determined `ZLM_JNIERR` is returned and `err` contains an error message.

```
int zlm_read_application_cert(JNIEnv *env, jobject activity, char *err);
```

Return the (oldest) application certificate as SHA-256. Before this method can be used the application ID has to be read once by calling `zlm_read_application_cert()`.

```
const char* zlm_application_cert(void);
```

**Determining the `ANDROID_ID` on a device** You can determine the `ANDROID_ID` on a device by:

- Either run the Android Debug Bridge: `adb shell 'settings get secure android_id'`
- Or install the **Device ID app** from Evozi and run it. The first entry (Android Device ID) is the `ANDROID_ID`. By tapping on it one can easily share it via various means.

Afterwards just add the 16 character hexadecimal string to the `"hostid"` field of a license file and sign it with `zlm_sign`.



### Important

Make sure to call `zlm_read_android_id()` once in the beginning of your software, otherwise ZLM will not recognize the `ANDROID_ID`!

---

**Determining the application ID** Every Android app has a unique application ID that looks like a Java package name, such as `com.example.myapplication`.

The application ID is defined with the `applicationId` property in your module's `build.gradle` file, see [Android documentation](#).

To set an application ID as "hostid" prefix it with "applicationID=". For example, for the application ID `com.example.myapplication` the "hostid" entry would be "applicationID=com.example.myapplication".

**Important**

Make sure to call `zlm_read_application_id()` once in the beginning of your software, otherwise ZLM will not recognize the application ID!

---

**Determining the signing certificate** Every Android app is signed with a signing certificate. To get the SHA-256 hash of the used signing certificate do one of the following:

- Click on the Gradle menu in AndroidStudio, expand the Tasks tree, and click on `android` and `signingReport`. Look for SHA-256 hash of the corresponding builds.
- Run: `keytool -list -v -keystore keystore_name -alias alias_name`. Look for SHA-256 hash of the key.

**Important**

Make sure to call `zlm_read_application_cert()` once in the beginning of your software, otherwise ZLM will not recognize the application certificate!

---

## 13 iOS Specifics

The distributions for iOS can only build `libzlm.a` with an existing key pair and they do not build the tools (like `zlm_sign`). To generate a key pair and to sign license files for iOS use another distribution (for example, the one for Mac OS).

To link `libzlm.a` into your iOS application you also have to link in the Foundation and UIKit frameworks (usually done by Xcode automatically).

Because the "hostid" on iOS is vendor specific you cannot use a generic tool like "zlmhostid" to retrieve it, but only with the API given below.

**iOS only APIs** Return the vendor specific unique key (a UUID conforming to RFC 4122 version 4) as a C-string. See [Apple Developer Documentation](#) for details.

If the result is `NULL` wait and get the value again later. This happens, for example, after the device has been restarted but before the user has unlocked the device.

The value in this property remains the same while the app (or another app from the same vendor) is installed on the iOS device. The value changes when the user deletes all of that vendor's apps from the device and subsequently reinstalls one or more of them. The value can also change when installing test builds using Xcode or when installing an app on a device using ad-hoc distribution.

```
const char* zlm_ios_hostid(char *err);
```

Return the unique bundle identifier (CFBundleIdentifier). See [Apple Developer Documentation](#) for details.

```
const char* zlm_ios_bundle_identifier(char *err);
```

---

**iOS specific temporary failure** Since retrieving the vendor specific unique key might fail temporarily (see above) one should make sure it is available **before** calling `zlm_license_get()`. Example code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "zlm.h"

int main(int argc, char *argv[])
{
    ZlmLicense *license;
    char err[ZLM_ERRBUF];
    unsigned int seconds = 1;

    /* create license object */
    if (!(license = zlm_license_new(err))) {
        fprintf(stderr, "error: %s\n", err);
        return EXIT_FAILURE;
    }

    /* make sure the vendor specific unique key is available */
    while (zlm_ios_hostid(err) == NULL) {
        /* temporary failure, sleep before retry */
        sleep(seconds);
        /* double number of seconds for next sleep round */
        seconds <= 1;
    }

    /* try to get license for product 'My Product' from the same directory as the
       executed binary and if that fails from the current directory */
    if (zlm_license_get(license, "My Product", "1.0", argv[0], ".", NULL, err)) {
        fprintf(stderr, "error: %s\n", err);
        zlm_license_free(license);
        return EXIT_FAILURE;
    }
    else
        puts("got license!");

    /* *****
       * run your code here *
       ***** */

    /* free license object */
    zlm_license_free(license);

    return EXIT_SUCCESS;
}
```

## 14 Tool Sources

The ZLM tools are just simple wrappers for the tool functions given in the ZLM API (see above Section 8):

### zlmgenkeys.c

```
#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

int main(int argc, char *argv[])
{
```



```
char err[ZLM_ERRBUF];
int had_err;

if (argc != 1) {
    fprintf(stderr, "Usage: %s\n", argv[0]);
    fprintf(stderr, "Generate private and public keys for signatures.\n");
    return EXIT_FAILURE;
}

/* generate keys */
had_err = zlm_genkeys(err);

if (had_err) {
    fprintf(stderr, "%s: error: %s\n", argv[0], err);
    return -had_err;
}

return EXIT_SUCCESS;
}
```

### zlmhostid.c

```
#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

int main(int argc, char *argv[])
{
    char err[ZLM_ERRBUF];
    int had_err;

    if (argc != 1) {
        fprintf(stderr, "Usage: %s\n", argv[0]);
        fprintf(stderr, "Print host ID on stdout.\n");
        return EXIT_FAILURE;
    }

    /* print host ID on stdout and write it to file zlmhostid.txt in CWD. */
    had_err = zlm_hostid_file(err);

    if (had_err) {
        fprintf(stderr, "%s: error: %s\n", argv[0], err);
        return -had_err;
    }

    return EXIT_SUCCESS;
}
```

### zlmkeyhash.c

```
#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

extern const unsigned char* zlm_get_public_key(int *keylen);

int main(int argc, char *argv[])
{
    char err[ZLM_ERRBUF];
    int had_err;

    if (argc != 1) {
```

```
    fprintf(stderr, "Usage: %s\n", argv[0]);
    fprintf(stderr, "Print hash of public key on stdout.\n");
    return EXIT_FAILURE;
}

/* print public key hash */
had_err = zlm_keyhash(zlm_get_public_key, err);

if (had_err) {
    fprintf(stderr, "%s: error: %s\n", argv[0], err);
    return -had_err;
}

return EXIT_SUCCESS;
}
```

### zlmsign.c

```
#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

extern const unsigned char* zlm_get_private_key(int *keylen);

int main(int argc, char *argv[])
{
    char err[ZLM_ERRBUF];
    int had_err;

    if (argc > 2) {
        fprintf(stderr, "Usage: %s [license_file_to_sign]\n", argv[0]);
        fprintf(stderr, "Sign the given license file (or read from stdin).\n");
        return EXIT_FAILURE;
    }
    else if (argc == 1)
        fprintf(stderr, "reading license file from stdin...\n");

    /* sign license file */
    had_err = zlm_sign_file(argv[1], zlm_get_private_key, err);

    if (had_err) {
        fprintf(stderr, "%s: error: %s\n", argv[0], err);
        return -had_err;
    }

    return EXIT_SUCCESS;
}
```

### zlmverify.c

```
#include <stdio.h>
#include <stdlib.h>
#include "zlm.h"

extern const unsigned char* zlm_get_public_key(int *keylen);

int main(int argc, char *argv[])
{
    char err[ZLM_ERRBUF];
    int had_err;

    if (argc > 2) {
```

```
fprintf(stderr, "Usage: %s [license_file_to_verify]\n", argv[0]);
fprintf(stderr, "Verify the given license file (or read from stdin).\n");
return EXIT_FAILURE;
}
else if (argc == 1)
    fprintf(stderr, "reading license file from stdin...\n");

/* verify license file */
had_err = zlm_verify_file(argv[1], zlm_get_public_key, err);

if (had_err) {
    fprintf(stderr, "%s: error: %s\n", argv[0], err);
    return -had_err;
}

return EXIT_SUCCESS;
}
```

## 15 Changelog

### CHANGES IN VERSION 1.23 (2020-04-28)

- license files can have optional "cpu" architecture entries
- "cpu" entries can produce new error code ZLM\_WRONGCPU
- zlmsign: fix bug where applicationID of a certain length is falsely rejected

### CHANGES IN VERSION 1.22 (2019-10-24)

- fix error messages for version mismatch
- Android: compile with -fPIC instead of -fPIE
- If the *patch* number of a "version" entry in a license file is not set it is considered to be ULONG\_MAX.

### CHANGES IN VERSION 1.21 (2019-10-02)

- the zlmsign tool correctly handles license file entries of the form "*major.minor.patch*" (in addition to the old form "*major.minor*").
- Android: builds have been switched to LLVM from NDK r20 (Android API level 16 for 32-bit builds and level 21 for 64-bit builds).

### CHANGES IN VERSION 1.20 (2019-09-30)

- "version" and "minversion" license file entries can have the new form "*major.minor.patch*" (in addition to the old form "*major.minor*").
- Linux: support for the MIPS platform has been added

### CHANGES IN VERSION 1.19 (2019-08-15)

- Android only: "hostid" can also be an application ID (prefixed with "applicationID=").
  - Android only: new "appcert" entry in license files can be used to bind licenses to signing certificates.
  - iOS only: "hostid" can also be a bundle identifier (prefixed with "CFBundleIdentifier=").
-

## CHANGES IN VERSION 1.18 (2019-07-25)

- iOS: fix missing symbol in distribution
- fix documentation bug for the `zlmhostid` tool
- improve error message if the environment variable `ZLM_LICENSE` is defined, points to a directory, and contains an invalid license file.
- JSON parser has been updated
- Windows: due to problems with VS2013 only the VS2015 distribution builds DLLs and contains .NET and Java bindings.
- Windows: ZLM can now handle license files in directory paths which contain non-ASCII characters (Unicode). In error messages such characters are converted to ? for maximum portability.

## CHANGES IN VERSION 1.17 (2018-01-15)

- `zlm_sign_license()` added to ZLM API

## CHANGES IN VERSION 1.16 (2017-12-17)

- license files can have optional "expiryinfo" entries
- allow the `ANDROID_ID` to be 15 characters long (may happen on a HTC One).

## CHANGES IN VERSION 1.15 (2017-09-29)

- the `zlm_sign` tool reads from stdin and writes to stdout if no license file is given as argument.
- the `zlm_verify` tool reads from stdin if no license file is given as argument
- ZLM now runs on iOS and "iOS" is a valid "os" entry
- the `zlm_sign` tool signs license files with "os" entries set to "iOS"
- iOS only: "hostid" must be an UUID (conforming to RFC 4122 version 4)
- iOS only: `zlm_ios_hostid()` added to ZLM API
- the `zlm_sign` tool signs license files with UUID (RFC 4122) "hostid" entries
- the ZLM library exports less (unused) symbols
- Ignore errors reading out MAC addresses, if the `ANDROID_ID` has been read with `zlm_read_android_id()` beforehand. This fixes a problem on Android devices where the deprecated reading of MAC addresses is not available anymore.
- Windows: the Visual Studio distributions are build with VS2013 and VS2015

## CHANGES IN VERSION 1.14 (2017-03-01)

- Android only: "hostid" can also be an `ANDROID_ID`
  - Android only: `zlm_read_android_id()` and `zlm_android_id()` added to ZLM API
  - Android only: `zlm_read_android_id()` can return new error code `ZLM_JNIERR`
  - the `zlm_sign` tool signs license files with `ANDROID_ID` "hostid" entries
  - Android releases are now build with NDK r13b instead of r13
  - Linux: add additional x86 distribution which was build with Clang. Use this version on older Linux systems if you get a linking error with GCC.
-

## CHANGES IN VERSION 1.13 (2016-11-18)

- the optional "userdata" entry in a license file can also be an *object* now, in previous versions only a *string* was possible

## CHANGES IN VERSION 1.12 (2016-10-07)

- host IDs for Android now also contain eth devices
- 64-bit version for Android ARM added
- the `zlm_sign` tool now converts MAC addresses to lowercase (as they are shown by the `zlm_hostid` tool). This allows to use MAC addresses retrieved with other tools directly (like `ifconfig`).
- typos in error messages were fixed
- `zlm_hostid_json_hashed()` output fixed
- Android releases are now build with NDK r13 instead of r11c, the 32-bit version targets API level 16 and the 64-bit version API level 21.

## CHANGES IN VERSION 1.11 (2016-08-30)

- ZLM now runs on FreeBSD and "FreeBSD" is a valid "os" entry
- the `zlm_sign` tool signs license files with "os" entries set to "FreeBSD"
- `zlm_license_version()` added to ZLM API

## CHANGES IN VERSION 1.10 (2016-04-20)

- ZLM now runs on Android and "Android" is a valid "os" entry
- the `zlm_sign` tool signs license files with "os" entries set to "Android"
- a possible problem with the clock set back detection on some platforms was fixed

## CHANGES IN VERSION 1.9 (2016-02-27)

- the `zlm_hostid` tool prints the host ID(s) of the computer in unhashed form again (reverting the change introduced in version 1.7). Using the host ID(s) in unhashed forms makes it easier to detect the abuse of trial versions by a user reusing the same machine.
- the `zlm_sign` tool now hashes the host ID(s) (with a salt) before signing them. This makes it harder to spoof MAC addresses. To sign the host ID(s) in unhashed form (old behavior) set the environment variable `ZLM_SIGN_WITHOUT_HASHING` to any value. You should do that, if you want the signed license files to be parsed by binaries using ZLM version 1.6 or earlier.
- `zlm_hostid_json()` was undeprecated
- `zlm_hostid_json_hashed()` was deprecated in favor of `zlm_hostid_json()`, but continues to work as before

## CHANGES IN VERSION 1.8 (2016-01-16)

- the `zlm_hostid` tool additionally writes its output to a file `zlm_hostid.txt` in the current working directory.
  - a prebuild `zlm_hostid` tool is now part of the ZLM distributions
  - `zlm_hostid_file()` added to ZLM API
  - `zlm_free()` added to ZLM API
  - Windows: the Visual Studio distribution builds two versions now (with the compile flags `/MD` and `/MT`, respectively)
-

## CHANGES IN VERSION 1.7 (2015-08-21)

- the `zlmhostid` tool now prints the host ID(s) of the computer in a hashed form with a corresponding salt. This makes it harder to spoof MAC addresses. It also prints "os" entries now. Old (unhashed) "hostid" entries can still be signed and are still valid.
- `zlm_hostid_json_hashed()` added to ZLM API
- `zlm_hostid_json()` was deprecated in favor of `zlm_hostid_json_hashed()` but continues to work as before
- license files starting with an UTF-8 byte order mark (BOM) can be parsed

## CHANGES IN VERSION 1.6 (2014-08-19)

- the documentation of the `zlm_license_check_x()` methods has been improved
- `zlm_license_not_expired()` added to ZLM API

## CHANGES IN VERSION 1.5 (2014-08-17)

- the `zlm_license_check_x()` methods have been improved
- `zlm_license_userdata_unescaped()` added to ZLM API

## CHANGES IN VERSION 1.4 (2014-05-08)

- Java bindings are contained in most distributions
- Windows: the Visual Studio distribution builds a Dynamic Link Library (DLL)
- Windows: the Visual Studio distribution contains .NET bindings
- `zlm_hostid_json()` added to ZLM API
- license files can have optional "os" entries
- "os" entries can produce new error code `ZLM_WRONGOS`

## CHANGES IN VERSION 1.3 (2014-04-28)

- Linux: ZLM works better with older libc versions
- Linux: the ZLM library is position-independent (compiled with option `-fPIC`)
- Windows: the ZLM library works better with older Visual Studio compilers

## CHANGES IN VERSION 1.2 (2013-09-18)

- license files can have optional "minversion" entries
- "minversion" entries can produce new error code `ZLM_MINVERSION`
- `zlm_version()` added to ZLM API

## CHANGES IN VERSION 1.1 (2013-08-13)

- license files can have optional "userdata" entries
- `zlm_license_userdata()` and `zlm_license_next()` added to ZLM API

## CHANGES IN VERSION 1.0 (2013-06-06)

- initial release
-