RLM Embedded Getting Started Guide

RLM v12.4

July, 2018

Contents

Welcome

Welcome to the Reprise License Manager (RLM), the newest license manager brought to you by the team who developed FLEX*lm* ®

About this Manual

This manual, the *RLM Embedded Getting Started Guide*, contains the step-by-step guide to get you starting using RLM Embedded. When you have finished this guide, you will have run sample licensed applications as well performed a simple integration of RLM into your own application (if you choose to do so). This manual does not describe the setup and operation of the Reprise activation software (Activation Pro). That is described in the *RLM Activation Pro Getting Started Guide*.

Introduction To RLM Embedded

The Reprise License Manager (RLM) allows a software vendor (ISV) to flexibly price and license their product(s) for delivery to their customers. At its most basic level, RLM Embedded allows an ISV to deliver fixed (node-locked) licenses to their customers.

What sets RLM apart?

RLM was designed from the start to emphasize ***openness***, ***transparency***, and ***simplicity***.

RLM is ***open*** because we publish the format of our license file, so that your customers can always examine and know what your license rights are.

RLM is ***transparent*** in the sense that we do not allow "back doors" which lead to unique behaviors from one ISV to another. In addition, we have removed policy from the application code, and placed it into the license key itself, so that your customers will be able to understand the license terms without having to understand your implementation.

RLM is ***simple*** because we place the ***policy in the license***(tm) and not encoded into multiple API calls.

Licensing 10,000 foot Overview

If you have used other license management products, you can skip this chapter. If you are new to license management, however, we have included an overview of how license management products operate.

First, a few definitions

Term	How used in this manual
license manager	a software component which keeps track of the right to use a software product
product	Your software
product name	The name used by the product to request it's license
license	The right to use a product, incorporated into a short text description. Referred to by the product name
check out	The act of requesting a license for a product
check in	The act of releasing the license for a product
node-locked (license)	A license which can be used only on a particular specified computer
floating (license)	A license which can "float" on a network, in other words, one which can be used by anyone who can access the license server (Note: RLM Embedded does not support floating licenses).
ISV	Independent Software Vendor, i.e., your company

License Manager Overview

License managers control the allocation of licenses to use software products. They do this by allowing a product to *check out* and *check in* a named license. The license manager keeps track of which users and computers can use these licenses.

Most license managers provide APIs with calls to control many of the aspects of licensing behavior. First-generation license managers (such as FLEXlm) took the approach of providing extremely complex APIs and internal license server options to control license policy, with relatively less control contained in the licenses themselves.

Unlike the first-generation license managers, the design philosophy of RLM is to preserve the simplicity of the system for both ISVs and License Administrators by avoiding all unnecessary options in the client library and moving as many of these options to the license file as possible, where they are visible and understandable by everyone.

In general, even when API calls are available to control it, license policy should be kept out of the application, and placed into the license itself. This makes for a more understandable licensing system for both ISVs and License Administrators. This results in much more standard behavior of application licensing from ISV to ISV. The Reprise team learned this the hard way when we supported thousands of customers in the past, and we applied these lessons to the design of RLM.

License Types

RLM Embedded supports node-locked licenses only. The full version of RLM supports:

- node-locked (runs on a specified node only)
- floating (available anywhere on a network, up to a concurrent usage limit)
- token or package-based

In addition, licenses will contain various attributes which further restrict their use. Some attributes are:

- expiration date
- highest available software version
- start date

RLM Embedded supports all the license attributes above, as well as many, many others.

License Manager Components

RLM embedded consist of 3 components:

- A client library
- License utilities, and
- A license description repository (i.e., a license file)

How To Get Licenses To Your Customer

Typically, licenses are delivered in text form to customers. Long ago, this was done via phone/fax/magnetic media. Today, the most common license delivery mechanism is the internet, either via email or automatic activation from an activation server at the ISV site.

RLM licenses are always 100% ascii text, and can be delivered by any convenient means, however email and activation are by far the most common delivery mechanisms.

Running the Demo - Quick-Start Guide

To get you started as quickly and easily as possible with RLM, we will first use the binaries on the kit. After you are familiar with the basic operation of RLM Embedded, you can integrate the RLM calls into your application.

[You should note that the RLM kit is the same for the full version of RLM and RLM Embedded. The functionality of the kit is controlled by the license you receive from Reprise Software. RLM-embedded licenses all contain the string "options=embedded".]

To start, we will download the kit, then the next 2 chapters will give detailed directions on how to build the kit for Windows and Unix/Mac systems.

Download the RLM kit from the Reprise website

To download RLM, go to the Reprise Website Download area, enter your username and password, and select the kit(s) you want to download. Save this on your system.

Note: When downloading Unix or Mac kits using Internet Explorer on Windows XP systems, the files are incorrectly named as 'platform.tar.tar', rather than 'platform.tar.gz', once downloaded. This is a browser issue - after transfer, please rename the file before installation.

Each kit has a descriptive name on the website. The file names of the kits follow Reprise Software's platform naming conventions with ".tar.gz" on Unix and Mac platforms. On windows, the kits are self-installing .exe files. Some examples are listed here. Reprise supports many more platforms than the ones listed here, so if you don't see your platform, contact your Reprise salesperson:

Platform	Platform Name	Kit file name
Linux on Intel X86	x86_l2	x86_l2.tar.gz
Mac on Intel X86	x86_m1	x86_m1.tar.gz
Windows on Intel X86 (visual C 2005, 2008)	x86_w2	rlm.v12.4BL1-x86_w2.exe
Windows on Intel X86 (visual C 2010, 2012)	x86_w3	rlm.v12.4BL1-x86_w3.exe
Windows on Intel X86 (visual C 2015 and later)	x86_w4	rlm.v12.4BL1-x86_w4.exe
Windows 64-bit on intel (Visual C 2005, 2008)	x64_w2	rlm.v12.4BL1-x64_w2.exe
Windows 64-bit on intel (Visual C 2010, 2012)	x64_w3	rlm.v12.4BL1-x64_w3.exe
Windows 64-bit on intel (Visual C 2015and later)	x64_w4	rlm.v12.4BL1-x64_w4.exe

Building the Demo on Windows

1. **Extract the kit files:**

   On Windows, run the installer from the .exe file you downloaded. The rlm files will be installed into the folder you selected (My Documents\Reprise\rlm.v12.4BL1-x86_w2 by default for the x86_w2 kit for RLM v12.4BL1).

2. Build the kit:

You have 2 options for configuring RLM on Windows - you can either use a Visual Studio or Visual C++ Project, or a Command Window. Each method has the same outputs; choose the method you're more comfortable with.

NOTE for VC2008 users:

The libraries in the x86_w2 and x64_w2 builds are created with VC2005. However, VC2008-compatible _md and _mdd variants are supplied. If you are using VC2008 and will be building executables that link dynamically to the C run time library (that is, you compile with /md or /mdd), you should put the VC2008-compatible libraries in place before you configure the SDK. Perform the following steps in the x86_w2 and/or x64_w2 folders:

1. Copy rlmclient_md.lib and rlmclient_mdd.lib to different names, in case you ever need them later for VC2005 builds.
2. Rename rlmclient_md90.lib to rlmclient_md.lib
3. Rename rlmclient_mdd90.lib to rlmclient_mdd.lib

To build using Visual Studio/Visual C++:

1. The platform directories (x86_w1, x86_w2, and x64_w2) contain Microsoft Visual Studio or Visual C++ project and workspace files. Double-click on the appropriate file to launch Visual Studio/Visual C++. In x86_w1, double-click on x86_w1.dsp. In x86_w2, double-click on x86_w2.vcproj. In x64_w2, double-click on x64_w2.vcproj.
2. When the development environment comes up, click on the Build menu and select "Rebuild All" (Visual C++) or "Build Solution" (Visual Studio). When the build is done, the output window should indicate 0 errors and warnings.

   The Visual C++ project is based on Visual C++ v6, and the Visual Studio project on Visual Studio 2005. If you're using a later version of the development environment, it will prompt you to allow it to convert the project to the later version. Allow it to do so, then proceed.

To build using a Command Window:

1. Create a command window with the Visual C++ environment set up
    - If your system has the Visual C++ environment set for command line use (this is common with Visual Studio 6) create a command window.
    - If your system doesn't have the Visual Studio environment set for command line use, you can either:
        - Create a command window and run a batch file provided by Microsoft to set up your command window for the next step. For 32-bit builds, the batch file to run is "vcvars32.bat"; for 64-bit builds it is "vcvarsamd64.bat". The location of these batch files is under "Program Files\Microsoft Visual Studio".

        -OR-

        - Create a command window via the Start->MS VisualStudioxxx or Start->MS Visual C++ menu. The specific sub-menu items vary with version but the target is "Visual Studio Comand Prompt".
2. cd to the platform directory of the SDK, for example

    ```
    cd x86_w3
    ```
3. Type nmake

Building the Demo on Unix or Mac

1. **Extract the kit files:**

   Use gunzip/tar to extract the archive:

```
% gunzip platform.tar.gz
% tar xvf platform.tar
% ./INSTALL  (enter a <CR> when prompted for an ISV name).
```

2. **Build the example program** from the *platform* directory (eg, x86_l2, x86_m1, etc):

   First, change directory to the place where you extracted the kit. Next, execute the following 3 commands:

```
% cd platform_dir (eg: x86_l2, x86_m1, sun_s1, etc)
% make
% rlmsign example.lic
```

Running the Demo

1. **Run the example program**

   Start in the *platform_directory* directory (the same directory you were in above), and run the *rlmclient* program to check out an *rlmclient* license. Run rlmclient as follows:

   ```
   % rlmclient
   ```

   The license you just checked out looks like this (this license is in the file *example.lic*):

```
LICENSE demo  rlmclient 1.0 permanent uncounted hostid=any sig=...
```

About this license:

- "demo" is the ISV name. When you purchase RLM Embedded, you will pick your ISV name.
- "rlmclient" is the license name. The rlmclient program checks out an "rlmclient" license by default.
- "1.0" is the license version. Any checkout of a version <= 1.0 will be satisfied by this license.
- "permanent" is the expiration date, ie, this license never expires.
- "uncounted" is count field of the license. "uncounted" means "uncounted, node-locked". An uncounted license must always have a hostid.
- "hostid=any" is the hostid where this license can be used. In this case, for the example, we created a license which works anywhere. In practice, you will get your customer's hostid at the time of purchase, and create a license for that host.
- "sig=..." is the license signature. This is inserted by the *rlmsign* program, or the license can be created and signed by RLM Activation Pro.

Congratulations! You have now built the sample program and checked out a node-locked license. There's not much more to it than that. Now, you are ready to integrate the RLM calls into your application and try it out. You can browse some example code in the *examples* directory on the kit (the example application is also located in *Appendix A – RLM Example Client Program*, on page 24.)

Integrating RLM Into Your Product

If you would like to integrate RLM into your own product, you will need to configure the RLM libraries and add calls to the RLM functions in your software.

If your application is written in Java, you should read this chapter to familiarize yourself with the basic RLM concepts. There are some Java-specific installation and integration instructions in the *RLM Reference Manual*. If you have a .NET application, you will find instructions for integrating RLM in the *RLM Reference Manual*.

Reprise Software recommends that you take a look at the rlmclient.c sample program in the *examples* directory of the RLM kit. This example shows the use of the first 7 of the 8 functions in the ***RLM core API.*** This example program is also contained in Appendix A of this document.

These first 7 functions are the basic functions you will use in your application. When you are ready to learn more about these or other RLM functions, consult the RLM Reference Manual.

The game plan

As an ISV you integrate RLM by adding calls from the RLM client library into your application. You then ship your product plus a few additional components of the RLM license system, as required. You can accomplish the engineering portions of these tasks in less than a day – the hardest work is deciding what to license, and what license rights to grant to your customers. Once you integrate RLM, the additional components you ship are:

- a license file to describe your customer's rights to the product (custom-generated by you for each of your customers)
- the rlm utilities (rlmutil – a standard part of the RLM kit)

Except for the license file, the components are the same for every one of your customers. The actual license file, which describes your customer's rights to the product, will (in almost all cases) be different for every one of your customers.

RLM embedded requires no network connection nor license server processes. If you use the optional RLM Activation Pro product, you will need network connectivity (over the internet) to your activation server.

Given that background, now we are ready to start.

Integrating RLM into your application – the 3 steps

To integrate RLM into your software, there are three steps:

1. Download and install the kit from the Reprise website
2. Configure your RLM libraries
3. Add RLM API calls to your application

These steps are described in the following sections.

- **Step 1: Download and install the kit from the Reprise website**

  If you ran the demo in the second chapter, you have already done this. If not, follow the instructions there.

- **Step 2: configure your RLM libraries**

  Windows:

  The kit is a self-installing .exe file. Run the installer, which will extract the kit directories (src and x86_w1 or x86_w2 for 32-bit or x64_w2 for 64-bit). By default these are installed into My Documents on your system.

  You have 2 options for configuring the libraries on Windows – you can either use a Visual Studio or Visual C++ Project, or a Command Window. Each method has the same outputs; choose the method you're more comfortable with.

  Configuring RLM with Visual Studio

  The platform directories (x86_w1, x86_w2, and x64_w2) contain Microsoft Visual Studio or Visual C++ project and workspace files. Double-click on the appropriate file to launch Visual Studio/Visual C++:

Platform	File to double-click
x86_w2	x86_w2\x86_w2.vcproj
x64_w2	x64_w2\x64_w2.vcproj
x86_w3	x86_w3\x86_w3.vcproj
x64_w3	x64_w3\x64_w3.vcproj
x86_w4	x86_w4\x86_w4.vcproj
x64_w4	x64_w4\x64_w4.vcproj

  Click on "Rebuild All" in the Build menu in Visual C++, or "Build Solution" in Visual Studio. The build will complete after a few seconds and the output window will indicate 0 errors and 0 warnings.

The Visual C++ project is based on Visual C++ v6, and the Visual Studio project on Visual Studio 2005. If you're using a later version of the development environment, it will prompt you to allow it to convert the project to the later version. Allow it to do so, then proceed.

Configuring RLM with a Command Window

To launch a Command Window with the development environment already set up, use one of the options in the:

Start->All Programs->Microsoft Visual Studio 200x->Visual Studio Tools

menu. The options differ with the specific version and edition of Visual Studio, but choose the one that does native development for the platform you're on. In other words, if you're on a 32-bit system, choose the option that does 32-bit development, and if you're on a 64-bit system, choose the options that does 64-bit development. This correct choice may not always be clear from the names of the options, but the command window that's launched will display a message at the top saying:

"Setting environment for using Microsoft Visual Studio 200x x86 tools."
or
"Setting environment for using Microsoft Visual Studio 200x x64 tools."

so you'll know if you've chosen the correct one.

If you are using older Visual C compilers and your system doesn't have the Visual Studio environment set for command line use, you need to run a batch file provided by Microsoft to set up your command window for the next step. For 32-bit builds, the batch file to run is "vcvars32.bat"; for 64-bit builds it is "vcvarsamd64.bat". The location of these batch files is under "Program Files\Microsoft Visual Studio". When you have set this up, open a command window.

Next, do the following:

```
C:> REM  cd to the place where you extracted the kit.
C:\your_kit> cd x86_w1  (or x86_w2)
C:\your_kit\x86_w1> nmake
```

Unix or Mac:

(Note: you have already performed these steps when you did the demo earlier)

```
% gunzip platform.tar.gz
% tar xvf platform.tar
% ./INSTALL
% cd platform
% make
```

All platforms:

RLM kits are pre-built for ISV "demo", with licenses that expire in 30-60 days after the RLM kit release date. If your demo license has expired, you will need to put the new license you received from Reprise Software into the file license_to_run.h in the src directory. If you have purchased RLM, you will need to edit license_to_run.h to replace the license there with your permanent license, and you will also need to edit the makefile in the binary directory (x86_w1, x86_w2, x64_w2, x86_l2, etc...) to change your ISV name.

If you are using Java, there are a few additional steps required. These are described in the *RLM Reference Manual.*

For the curious, the detailed contents of the RLM kit are contained in *Appendix B - RLM Kit Contents* on page 26.

● **Step 3: Add RLM API calls to your application**

Using the example rlmclient.c as a guide, add the RLM api calls to your application. You will need rlm_init() and rlm_checkout() calls at a bare minimum, however it is good practice to call rlm_checkin() when you are finished with the license, and rlm_close() if your program makes no further licensing calls

Once you have done this, compile your application using the RLM include files in the directory <kit_dir/src>, and link with the RLM client library.

You're done!

Now that your application has been built with the RLM calls included, copy the example license file and edit it with your product names, etc. Use the *rlmsign* utility to sign the license file, and experiment with some node-locked and floating licenses. Don't forget to read the note on your public-private key pair below.

An important note on your public-private key pair

Step 2, above, created a public-private key pair for you as part of the *make* (or *nmake*) command, or build in Visual Studio. Before you use RLM in your product, you need to create a *public-private key pair that you will use for all your licenses, and* ***you should do this only one time.*** The key pair will affect the licenses you create, and you want to be able to process older license keys with newer versions of your software. Note that you should do this once, **not** once per platform you install.

You can safely ignore the remainder of this note for now, but you should return to this and understand the implications before you begin your RLM production implementation.

To create your key pair, run the *rlmgenkeys* utility. *rlmgenkeys* creates a pair of files:

- rlmpubkey.c - your public key - this gets built into your application
- rlmprivkey.c - your private key - this gets built into *rlmsign* to create your license keys

To run *rlmgenkeys*:

```
% cd kit-dir
% cd src
% ../platform-dir/rlmgenkeys
```

Where:

- kit-dir is the directory where the RLM kit resides, and
- platform-dir is the RLM binary directory for the machine on which you are running.

If you do not share *src* directories on your various platforms, run *rlmgenkeys* once and copy the resulting files to all the other *src* directories you use. Once you have created your key pair and installed it in the src directories in all your RLM kits, do a "make" in each kit to update the rlm.a library.

You should be ***very careful*** with these two files. **DO NOT LOSE THEM**. **Do not allow your private key file (or *rlmsign*) outside your company**. If your private key file (or *rlmsign*) becomes compromised, others will be able to make licenses for your products. Once you generate these files, you should copy them to a safe place where they will not be lost, and where they will be secure.

When you upgrade to a newer version of RLM, you will be asked for the location of these two files, so that the new version will generate compatible keys with your older versions.

Making Your Product Production-Ready

You now know how to integrate RLM into your product. This chapter describes considerations for a good RLM implementation.

If you are evaluating RLM, you can skip this chapter for now. You will want to return to it later for guidance on how to make your product ready to ship.

Productizing your licensing implementation

In order to make your license management production-ready, there are 4 main steps:

1. Decide on your Licensing Strategy
2. Configure your RLM libraries with your permanent options
3. Package your software for shipment
4. Prepare to create licenses for your customers

These steps are described in the following sections.

● Step 1: Decide on your Licensing Strategy

RLM allows you to request and release *licenses* for *products*. The *license* for a product has certain attributes, which are described in the license grant itself (which is contained in the license file). The most basic license attributes are:

- ISV name (you pick this when you purchase RLM)
- Product name
- Highest Version supported
- the node identification, since all RLM Embedded licenses are node-locked.
- Expiration date

Before you integrate RLM into your application, you must decide which products you wish to license and select the *product* names for the licenses. It is generally recommended that you choose names that correspond very closely to the name which your customer purchases - it makes license administration much more straightforward for your customers if the name of the *product* in the license is the same as what they purchased. Note that the *product* name must be less than 40 characters.

In addition, each license request will specify a *version*. The two main strategies for selecting versions are either (a) make the version number match the major version of your software, in which case a new license would be required by your

customers for each major release of your product or (b) only change the version in the license request occasionally, when you want to force your customers to purchase a new license.

So, before you start to integrate the code into your application, you should decide:

- Where do you want to request and release licenses
- What is the name of the license(s)
- What license version to request.

(Note: There is more information about these issues in the chapter on Creating Licenses.)

Generally, the first two decisions will stay the same over the life of the software product, while you will update the license checkout version from time to time.

- **Step 2: Configure your RLM libraries with your permanent options**

There are 4 configuration items you must complete before you build your RLM kit:

- Install your permanent RLM license into *license_to_run.h*
- Create your public/private key pair, which is done one time only (See the note in the last chapter.)
- Configure your RLM parameters in the file *rlm_isv_config.c*
- Modify the makefile to change the ISV name "demo" to your permanent ISV name.

To install your RLM license, edit the file src/license_to_run.h, using the parameters you received in the email from Reprise Software. (Note: RLM kits are pre-built with demo license keys which expire in approximately 2 months from the date of kit release, so you may be able to skip this step if you are evaluating RLM).

Your applications are built from components supplied by Reprise Software. You need to provide 2 custom inputs for the build:

- Your Public Key, for license key verification - rlm_pubkey.c - (This was done in step #2, above. See Create your Keys).
- A file of RLM customizations called rlm_isv_config.c (this file is contained in the **src** directory on the kit)

rlm_pubkey.c is created by the rlmgenkeys utility. You should run this **only once** to create your public/private key pair. Once you create these files, save them - if you lose one of these files, you will no longer be able to generate license keys compatible with older versions of your software.

rlm_isv_config.c contains calls to:

- set up your ISV name
- install your RLM license (do not change this call)
- register ISV-defined hostids, and
- include or exclude code for optional hostids (e.g., dongles, etc)

Edit this file before compiling your applications.

Once you have created these 2 files you are ready to link your applications with the RLM libraries.

● Step 3: Package your software for shipment

With RLM, you specify nearly all licensing options in the actual license that you ship to your customers. However, there are a few issues that you need to consider before you ship your application:

- Review the RLM API calls you make in your application to be sure that you use product names that are suitable (we strongly recommend using the name of the product that is in general use), and that the version numbers are correct. If you intend for your customers to be able to use old licenses from your product, be sure that the version number in the *rlm_checkout()* call is appropriate.
- If we have provided you with special debug libraries, make sure you use the non-debug libraries from the standard kit for your release.
- Ensure that you have included the RLM License Administration Tools in your distribution kit.
- Review the Best Practices for RLM Integration section and ensure that your product and installation are well-behaved.

Reserved Product Names

In general, your product names need only be unique to your company. However, any product name beginning with the 4 characters "rlm_" is reserved.

● Step 4: Prepare to create licenses for your customers

You will want a system in place to fulfill licenses for your customers before you ship your product. Review the Creating Licenses chapter on page 22 to chose the way you will do your license fulfillment.

Using RLM with the Visual Studio GUI

If you use the Visual Studio GUI interface on Windows, the procedure to configure the RLM libraries is as follows:

- In a command window, build the RLM SDK as specified in Installing RLM. You need do this only once per release of RLM.

- In your project settings / properties in Visual Studio:

    - Under C/C++, add **<RLM SDK path>\src** to the Additional Include Directories (where **<RLM SDK Path>** is the path to the installed RLM SDK)

    - Under the Link/Input/Additional Dependencies or Additional Library Path, add **<RLM SDK path>\<platform>\rlmclient.lib** (where **<platform>** is either **x86_w1, x86_w2** or **x64_w2**, depending on whether the project is a 32- or 64-bit project)

    - Under the Link Command Line or Project Options section, make sure the following libraries are included:

        - ws2_32.lib
        - Advapi32.lib
        - Gdi32.lib
        - User32.lib
        - winhttp.lib
        - netapi32.lib
        - kernel32.lib
        - oldnames.lib
        - shell32.lib
        - libcmt.lib

Then you will be able to use RLM in your project without leaving the GUI.

Using RLM with .NET - See the *RLM Reference Manual* for detailed instructions.

Using RLM with Java - See the *RLM Reference Manual* for detailed instructions.

Best Practices for RLM Integration

Our experience supporting thousands of FLEX*lm* ISVs and License Administrators has taught us that certain design decisions can cause long-term support problems. While we have made every effort to remove options from RLM which cause License Administrator confusion with little corresponding benefit, there are still things that you can do to make things easier for your customer's installation and support.

In this section, we attempt to provide a framework for how *well-behaved* applications use RLM. Adherence to these guidelines, while not strictly mandatory, will be greatly appreciated by your customer's License Administrators who will see more consistent implementations from ISV to ISV. This will also translate into support savings for you, as applications from different ISVs will behave in a more consistent fashion.

Product names

The name you use to check out a license for a product should be as close to the name of the product you sell as possible. Fewer checkouts per product are generally better from an License Administrator support and understanding standpoint. In the early days of license management, companies literally "went crazy" adding checkout calls to smaller and smaller pieces of their application, which resulted in several licenses required to run one product. Resist the temptation to do this. If your product is a schematic editor, you probably don't need checkout calls to license the code that reads and writes the data files. You might, but probably not.

Reprise Software considers it best practice to:

Use the name from your price list in the *rlm_checkout()* call, or a name as close to this as possible.
Use as few *rlm_checkout()* calls as possible to accomplish your licensing strategy. Why? See Use Few Checkout Call, below
AVOID THE USE of license text fields (such as customer, contract, etc) to control how your application behaves, other than presenting this data to the user.
DO NOT USE the *rlm_license_xxxx()* calls (other than rlm_license_stat()) to do anything beyond displaying information to your user.

Installation of your product and finding the licenses for it to operate

When you integrate RLM into your product there are issues concerning delivery of your product and the licenses for it to operate. As you already know from the chapters on Integrating RLM Into Your Product, and The License File, there are a few ways that your application can locate the licenses it needs to operate:

- RLM_LICENSE (or <ISV>_LICENSE) environment variable
- options you provide to your user to specify a license location, and
- licenses present in your product's binary directory

Reprise Software considers it best practice to:

AVOID **using RLM_LICENSE or <ISV>_LICENSE** as part of your installation scripts or adding definitions of these variables to your user's environment. If you want to set a default license file, you should do this by locating the license file (or a link to the license file) in the directory with your binaries, or by using the optional license location in the first parameter to *rlm_init()*.
ALWAYS **leave RLM_LICENSE and <ISV>_LICENSE environment variables unset** - so the License Administrator can override any defaults you have specified.
ALWAYS **provide the path to your binary as the second parameter to *rlm_init()***. In this way, your customer's License Administrator will know that they can put the license file (or a link) in this directory and it will be the "last resort" license file to be used.

Use Few Checkout Calls

The recommendation to use as few checkout calls as possible is made in response to our experience in talking with many end users. In general, the more fragmented into separate license domains an application becomes, the less end users understand the licensing behavior and the less satisfied they are. In an ideal world (from the end user's point of view), an application would need to check out 1 license in order to run,and the name of that license would be the name of the application.

In practice, it's often quite reasonable for ISVs to use multiple license names in an application - just keep it within reason. A good rule of thumb is to use distinct licenses for things you charge extra money for. It seems obvious, but many ISVs have gone far, far beyond that - to the dissatisfaction of their customers.

Creating Licenses

When you ship your product to your customers, it will require a license to run. Generally, you want to grant different license rights to each customer. In order to do that, you create a unique *license file* for each customer.

The license file consists of lines of readable text which describe the actual license grants to your customers. For a complete description of the license file format, see the *RLM Reference Manual*.

There are three main ways to create and ship licenses:

- standard *rlmsign* utility
- custom license generator built with *rlm_sign_license()* API call
- RLM Activation Pro

rlmsign – the standard License creation tool

RLM Embedded is shipped with a license creation tool called ***rlmsign*** which can be integrated into your fulfillment process. This tool reads a template license file and computes the *license key* for each license contained in the file. This license key authorizes the license and prevents tampering with the license parameters.

Using rlmsign:

```
rlmsign license_file [bits-per-character]
```

rlmsign reads *license_file*, computes the license keys for all the included licenses that specify your ISV name, and re-writes the file with the updated license keys.

The optional parameter *bits-per-character* is one of 4, 5, or 6, and specifies the character encoding of the resulting license key. If not specified, *bits-per-character* defaults to 5.

• *bits-per-character* of 4 results in license keys consisting of hexadecimal numbers only. The resulting key is approximately 92 characters in length.
• *bits-per-character* of 5 (the default) results in license keys consisting of uppercase letters and numbers only. The resulting key is approximately 74 characters in length.
• *bits-per-character* of 6 results in license keys consisting of upper and lowercase letters, numbers, and the 4 special characters ('*', '=', '+', and '~'). The resulting key is approximately 62 characters in length.

License creation API – *rlm_sign_license()*

In some cases, it is more convenient to build the license in-memory and sign that license directly before it is written to a file. In general, it is better to create the licenses in a file and use ***rlmsign*** to sign the licenses, however an API call is available for cases where this is not practical.

RLM has the *rlm_sign_license()* API call to sign a license line in-memory. For details on the *rlm_sign_license()* API call, see the *RLM Reference Manual*.

RLM Activation Pro

RLM Activation Pro is an optional add-on to RLM which allows you to give your customer an *activation key* which then allows your customer to retrieve their license from your website at a later time. The *activation key* is a short string (resembling a credit-card number) which can be generated in advance. Once the customer knows the system where they wish to use the software, the RLM activation software creates the license and transmits it to the user, creating the license file for them. Details of RLM Activation Pro are in the *RLM Activation Pro Manual*.

Should your activation needs exceed the capabilities of RLM Activation Pro, Reprise Software recommends a relationship with one of our License Fulfillment Partners. See our website Partner Page for more information on our Fulfillment Partners.

Reserved Product Names

In general, your product names need only be unique to your company. However, any product name beginning with the 4 characters "rlm_" is reserved, and should not be used by you for your products.

Appendix A – RLM Example Client Program

This example program (rlmclient.c) is contained on the RLM kit in the *examples* directory. Use this as an example of how to use the RLM API calls.

```
/*****************************************************************************

           COPYRIGHT (c) 2005, 2018 by Reprise Software, Inc.
        This software has been provided pursuant to a License Agreement
        containing restrictions on its use.  This software contains
        valuable trade secrets and proprietary information of
        Reprise Software Inc and is protected by law.  It may not be
        copied or distributed in any form or medium, disclosed to third
        parties, reverse engineered or used in any manner not provided
        for in said License Agreement except with the prior written
        authorization from Reprise Software Inc.

 *****************************************************************************/
/*
 *      Description:   Test client for LM system
 *
 *      Usage:         % sampleclient [product [count [version]]]
 *
 *      Return:        None
 *
 *      M. Christiano
 *      11/27/05
 *
 */

#include "license.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifndef _WIN32
#include <unistd.h>
#include <strings.h>
#endif /* _WIN32 */

static void printstat(RLM_HANDLE, RLM_LICENSE, const char *);

#ifdef x64_v1 /* No main on vxworks downloadable kernel modules, or argc/argv */
int sampleclient(char *product)
{
  char *argv[1];
        argv[0] = product;
        return main(1, argv);
}
#endif

int
main(int argc, char *argv[])
{
  RLM_HANDLE rh;
  RLM_LICENSE lic = (RLM_LICENSE) NULL;
  int x, stat;
  char *product, p[RLM_MAX_PRODUCT+1];
  int count = 1;
  const char *ver = "1.0";

        rh = rlm_init(".", argv[0], (char *) NULL);
        stat = rlm_stat(rh);
        if (stat)
        {
          char errstring[RLM_ERRSTRING_MAX];
```

```
                (void) printf("Error initializing license system\n");
                (void) printf("%s\n", rlm_errstring((RLM_LICENSE) NULL, rh,
                                                        errstring));
        }
        else
        {

/*
 *              Use the program name as the license name
 */
                if      ((product = strrchr(argv[0], (int) '/')))  product++;
                else if ((product = strrchr(argv[0], (int) '\\'))) product++;
                else                                    product = argv[0];
                strncpy(p, product, RLM_MAX_PRODUCT);
                p[RLM_MAX_PRODUCT] = '\0';
/*
 *              Don't want .exe
 */
                if ((product = strchr(p, '.'))) *product = '\0';
                product = p;
/*
 *              If product name specified, override program name
 */
                if (argc > 1) product = argv[1];
                if (argc > 2) count = atoi(argv[2]);
                if (argc > 3) ver = argv[3];
                lic = rlm_checkout(rh, product, ver, count);
                printstat(rh, lic, product);
        }

        (void) printf("Enter <CR> to continue: ");
        x = fgetc(stdin);

        if (lic)
        {
#if 0
/*
 *              rlm_checkin() isn't necessary if you aren't going
 *              to do anything else on the handle (other than check
 *              in licenses).   If you are using a handle created
 *              with rlm_init(), then rlm_checkin() doesn't hurt
 *              anything.  But if you use a handle created with
 *              rlm_init_disconn(), then rlm_checkin() causes an extra,
 *              unnecessary network connection to the license server.
 */
                rlm_checkin(lic);
#endif
                rlm_close(rh);
        }
        return(stat);
}

static
void
printstat(RLM_HANDLE rh, RLM_LICENSE lic, const char *name)
{
  int stat;
  char errstring[RLM_ERRSTRING_MAX];

        stat = rlm_license_stat(lic);
        if (stat == 0)
                (void) printf("Checkout of %s OK.\n", name);
        else if (stat == RLM_EL_INQUEUE)
                (void) printf("Queued for %s license\n", name);
        else
        {
                (void) printf("Error checking out %s license\n", name);
                (void) printf("%s\n", rlm_errstring(lic, rh, errstring));
        }
}
```

Appendix B - RLM Kit Contents

Each RLM kit (for a particular platform) is contained in 4 subdirectories:

- Machine-independent subdirectory (*src*)
- Machine-independent examples subdirectory (*examples*)
- Evaluation directory – pre-built binaries (*eval*)
- Machine-dependent subdirectory (name varies for each platform)

The Machine Independent (*src*) directory contains:

File	Contents
license.h	rlm include file
license_to_run.h	License for RLM itself
rlm_admin.h	Admin API include file (optional product)
rlm_isv_config.c	Configuration data for RLM
RELEASE_NOTES	Release notes for this version of RLM
RLM_Reference.txt	Pointer to RLM documentation on website
VERSION	RLM kit version information

The Machine Independent (*examples*) directory contains:

File	Contents
act_api_example.c	Sample client-side activation code
activation_example.html	Sample HTML page for activation
actpro_demo.c	Demo program for activation pro
detached_demo.c	Sample code to implement a detached demo
example.opt	Example License Administration option file (UNUSED in RLM Embedded)
integrate_older.c	Example code for integrating RLM alongside an older LM
isv_hostid_example.c	Example rlm isv-defined hostid
rehost_example.c	Example program using a rehostable hostid
rlm_transfer.c	Example ISV-defined license transfer code (UNUSED in RLM Embedded)
rlmclient.c	Example rlm application program
roam_example.c	Example code to implement license roaming (UNSED in RLM Embedded)
unsupported/	Various unsupported examples

Each Unix/Mac Platform-dependent directory contains (before executing "make"):

File	Contents	Notes
example.lic	Example license file	Created by INSTALL
librlm.a	Symbolic link to rlm.a	
makefile	Makefile	
rlm	The generic rlm server	UNUSED
rlm.a	RLM library	
rlmanon	RLM logfile anonomizer	UNUSED
rlmmains.a	RLM main() functions for misc. programs	
rlmutil	RLM utilities	

The Windows Platform-dependent directory contains (before executing "nmake"):

File	Contents	Notes
example.lic	Example license file	
isv_main.obj	main() for ISV server	UNUSED
isv_server.lib	library for ISV server	UNUSED
makefile	Makefile	
rlc.obj	main() for Activation administration (rlc)	
rlm.def	RLM DLL export definitions	
rlm.exe	The generic rlm server	UNUSED
rlm.res	RLM version resource file	
rlm_genlic.obj	License generator object	
rlm_mklic.obj	main() for Activation license generator	
rlmact.obj	rlc object file	
rlmanon.exe	RLM logfile anonomizer	UNUSED
rlmclient.lib	RLM client library	
rlmclient_md.lib	RLM client library - compiled with /Md	
rlmclient_mdd.lib	RLM client library - compiled with /Mdd	
rlmclient_mtd.lib	RLM client library - compiled with /Mtd	
rlmgen.obj	rlc license generation module	
rlmgenkeys.obj	main() for rlmgenkeys utility	
rlmsign.obj	main() for rlmsign utility	
rlmutil.exe	RLM utilities	
rlmverify.obj	main() for RLM log file authentication utility	UNUSED
x*_w2.vcproj, x86_w1.dsw	Visual Studio/Visual C++ project for configuring the kit	

The platform names for RLM follow the convention:

arch_[os][ver]

where:

- *arch* is the Reprise Software name for the processor/chip architecture
- *os* is the Reprise Software identifier for the operating system, and
- *ver* is the Reprise Software identifier for our version of rlm OS support (note: this is NOT the operating system version)

For example, x86_w2 refers to Windows on x86 architecture, with Visual Studio 2005/2008 support. ppc_m1 refers to Mac OS on PPC architecture.

The Java directory (java_unix, java_win) contains:

File	Contents
doc	Directory of HTML documentation
makefile	Makefile
rlmVVRB.jar	Java Library (VV=ver, R=rev, B=build)
RlmClient.java	Example rlm application program
rlmjava.def	JNI DLL exports (Windows only)
INSTALL	Java kit installation script (Unix only)
VERSION	RLM kit version information

The dotnet directory (RLM .NET support) contains:

File	Contents
Reprise	Visual Studio 2005 Project Directory for RLM .net support
RLMTest	Visual Studio 2005 Project Directory for RLM .net Test program

Appendix C - RLM Hostids

RLM supports several different kinds of identification for various computing environments, as well as some generic identification which are platform-independent.

RLM's host identification (hostid) types are:

hostid type	meaning	example	Notes
ANY	runs anywhere	hostid=ANY	
DEMO	runs anywhere for a demo license	hostid=DEMO	
32 (or long)	32-bit hostid, native on Unix, non X86 based platforms	hostid=10ac0307	On Windows, the 32-bit hostid is the Disk Serial Number
ip (or internet)	TCP/IP address	hostid=ip=192.156.1.3	always printed as "ip="
disksn	Disk hardware serial number	hostid=disksn=WD-WX60AC946860	Windows only
ether	Ethernet MAC address	hostid=ether=00801935f2b5	always printed without leading "ether="
uuid	BIOS uuid	hostid=uuid=699A4D56-58BF- 1C83-D63C-27A8BEB8011A	Windows only
user	User name	hostid=USER=joe	
host	Host name	hostid=host=melody	

Note that since you will always be using these hostids on a LICENSE line, they will always be preceded by “hostid=”. The actual hostid is the part after “hostid=”.

To determine the hostid of a machine, use the hostid type from the table above as input to the *rlmhostid* command:

rlmutil rlmhostid *hostid type*

For example:

rlmutil rlmhostid long
or
rlmutil rlmhostid internet

Note: IP address hostids can contain the wildcard ('*') character in any position to indicate that any value is accepted in that position.

Appendix D – RLM Version Comparison

The RLM-embedded product provides nodelocked license capability only (ie, no license server). This means that any license models supported by the license server are not available. The following table summarizes the features that are available in each version.

	RLM Full Version	RLM Embedded
License Models		
Nodelocked, uncounted	x	x
Nodelocked, single	x	x
Nodelocked, counted	x	
Floating	x	
License delivery		
Browser-based license generator	x	x
Internet activation	Option	Option
Advanced Features		
License hold/minimum checkout time	x	
License issue date	x	x
License roaming	x	
License version	x	x
Named-user licenses	x	
User-based, host-based licenses	x	
License features/options control	x	x
Cloud computing support	x	
License platform restrictions	x	x
Replacement licenses	x	x
Upgrade licenses	x	
License start dates	x	x
License expiration dates	x	x
TerminalServer/VM support	x	x
License soft limits (overdraft)	x	
License sharing	x	
License timezone restrictions	x	x
Detached Demo Licenses	x	x