RLM Embedded Reference Manual

RLM Embedded v15.0

May, 2022

RLM Embedded Reference Manual

V15.0

May, 2022

Table of Contents

RLM Embedded Reference Manual

Reprise License Manager ™

Section 1 – License Management Introduction

This section of the manual contains the information which applies to most license managers. If you are new to License Management, we suggest you review these two chapters first.

Introduction to License Management

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.

The purpose of a license manager is to allow a software vendor (ISV) to flexibly price and license their product(s) for delivery to their customers. At their most basic level, license managers like RLM Embedded allow an ISV to deliver concurrent-use (floating) or fixed (node-locked) licenses to their customers. In the case of node-locked licenses, no license server is needed with RLM Embedded and other advanced license managers. Most license managers offer many other license types for delivery to customers, and these vary from license manager to license manager. The next chapter, License Models, describes the various way you can license your software with RLM Embedded.

License managers differ from Copy Protection because license managers give advantages to the ISV's customers as well. License managers allow your customer's organization to know that they are using purchased software within the license limits set by you, their ISV. In addition, license managers collect usage information (at the customer's option) for later reporting and analysis. If your license manager is open and transparent, this usage information is provided in a fully-documented report log format.

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
license server	Part of the license manager which controls access to licenses. The License Server is an optional component, typically only required when floating licenses are used
ISV	Independent Software Vendor, i.e., your company

Hostid	An identification for a particular computer used by the license manager to either node-lock a particular application, or to specify where the license server can run.

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, and, if the license is a *floating license*, the license manager keeps track of how many copies of the license are in use.

Most license managers provide APIs with calls to control many of the aspects of licensing behavior. In addition, license managers provide license administration options to control the behavior of the license servers. These options are specified in server option files or via command-line or web-based administration tools.

First-generation license managers 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 Embedded is to preserve the simplicity of the system for both ISVs and license administrators by avoiding all unnecessary options in the client library and the license servers 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, it is good practice to keep license policy out of the application and the license server, and place it into the license itself, to the extent that the license manager allows this to be done. 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 Embedded.

License Types and Attributes

Commercial license managers will allow an ISV to control the use of their licenses using various License Types. The most popular license types are:

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

Another common license type is ***metered*** (i.e. a limited number of executions or limited time of execution).

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

- expiration date
- highest available software version
- start date
- named-user (i.e., the license can only be used by a particular user)
- allowed platform for the application.

License Manager Components

Most commercial license managers consist of 3 components:

- A client library (or wrapper)
- A license server, and
- A license description repository (typically a license file)

RLM Embedded is similar in structure to most popular license managers. RLM Embedded uses the client library, rather than the wrapper approach. The RLM Embedded license servers consist of a pair of servers – the generic rlm server along with an ISV-specific server. Finally, RLM Embedded uses a license file as the repository for license descriptions.

While some license managers require the license server in all cases, RLM Embedded node-locked licenses do not require a license server – only your application and the license file.

How To Deliver Licenses To Your Customer

Typically, licenses are delivered in text form to license administrators. 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 Embedded 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.

License Models

In the previous chapter we talked about *License Types* and *Attributes*. The *license types* and *attributes* which are supported by your license manager are the building blocks which you use to create the *License Models* your company will use. These *License Models* are what you will then use to price and deliver your software.

The most important thing you will do when selecting a license manager is to pick one with a sufficiently rich set of license types and attributes in order to allow you to create the License Models you need, not only for today, but for the future. If your license manager isn't sufficiently flexible, then your marketing department will become increasingly frustrated because they will not be able to offer your software with the terms and conditions which can lead to increased sales.

What exactly is a *License Model*?

Put most simply, a *License Model* is a set of terms and conditions which your license manager enforces. For any given set of terms and conditions (i.e., the particular *License Model*), your company has a set of pricing guidelines for the sale of the product.

Let's take a simple example. Your company may sell your software in two different ways – a floating (concurrent use) license for $3000, and a node-locked license for $1200. In this case, the *License Model* could be called either floating or node-locked. (Note that floating licenses are supported by RLM, but not by RLM Embedded).

It is important to your development organization that changes to the license model do not result in code changes. This is also important to sales and marketing, who will want to try different License Model offers without having to wait for a new software release.

We will discuss a few common License Models in the remainder of this chapter, and with each one, we will list the RLM Embedded license attributes that are used to implement it. Don't worry if you do not understand the RLM Embedded nomenclature, this will make more sense after you read the chapter describing The License File on page 39.

Node-locked License

A node-locked license is a license grant which allows the software to be used on a particular computer, and on that computer only. Most typically, this license is uncounted, meaning that if the software is running on the specified computer, any number of instances are allowed to execute.

In RLM Embedded ⇨ set the *count* field of the license to "uncounted" or "0", and specify the *hostid* of the computer in the actual license. Typically, node-locked uncounted licenses do not require a license server, so they are very simple to deploy.

Node-locked, Single Use License

A variant of the node-locked uncounted license, it is sometimes desirable to allow only a single instance of the software to run on a particular computer.

In RLM Embedded ⇨ set the *count* field of the license to ***single***, and specify the *hostid* of the computer in the actual license.

"Maintenance-Thru-Date" License

Many ISVs wish to issue a license to their customer which allows the customer to run (forever) any version of the software which is released through a particular date, e.g. one year into the future. If the ISV releases a new version in 11 months, the customer can use this version as well, but no version which is released more than 12 months later. This is accomplished by what we call a "date-based" version.

In RLM Embedded ⇨ set the *version* field of the license to ***a date, in the format yyyy.mm***, and specify the release date in your call to rlm_checkout() in the same format. When you issue licenses, issue them with a version number corresponding to the expiration of their support. So, for example, if you want to issue one-year supported licenses, in May of 2013, you would issue licenses of version 2014.05. When you release your software in December of 2013, you would request version 2013.12 Note that while it is possible to use other date formats, the format above is used by RLM Activation Pro.

Section 2 – RLM Embedded Basics

This section of the manual contains the information you need to license and deploy your application using the RLM Embedded license manager.

Welcome

The RLM documentation is divided into 7 manuals:

Standard RLM Components

- *RLM Embedded Getting Started Guide* - an introduction to the basic concepts of license management and RLM Embedded
- *RLM Embedded Reference Manual* **(this manual)**- the complete reference to RLM Embedded
- *RLM Getting Started Guide* - an introduction to the basic concepts of license management and RLM
- *RLM Reference Manual* - the complete reference to all core RLM components
- *RLM License Administration Manual* - The License Administration manual, suitable for shipment to your customers

Optional RLM Components

- *RLM Activation Pro Getting Started Guide – An Introduction to the RLM Activation Pro software*
- *RLM Activation Pro Manual* - Reference for the Optional RLM Activation Pro software

All seven manuals are available at the Reprise Website:

http://www.reprisesoftware.com/kits/RLM_Getting_Started_Guide.pdf
http://www.reprisesoftware.com/kits/RLM_Embedded_Getting_Started_Guide.pdf
http://www.reprisesoftware.com/kits/RLM_Reference.pdf
http://www.reprisesoftware.com/kits/RLM_Embedded_Reference.pdf
http://www.reprisesoftware.com/kits/RLM_License_Administration.html
http://www.reprisesoftware.com/kits/RLM_Activation_Pro_Getting_Started_Guide.pdf
http://www.reprisesoftware.com/kits/RLM_Activation_Pro.pdf

Integrating RLM Embedded into your product

As an ISV you integrate RLM Embedded by adding calls from the RLM Embedded client library into your application. Only if you plan to ship concurrent-use (floating) licenses will you also configure and build a license server. You then ship your product plus a few additional components of the RLM Embedded license system, as required. You can accomplish the engineering portions of these tasks in less than an hour – the hardest work is deciding what to license, and what license rights to grant to your customers. Once you integrate RLM Embedded, the additional components you ship are:

- a license file to describe your customer's rights to the product (custom-generated for each of your customers)
- the rlm utilities (rlmutil) provided by Reprise Software.

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.

When deployed to support node-locked licensing, no network connection nor license server processes are needed.

What sets RLM Embedded 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 you, or your license administrators can always examine and understand licensing activity.

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 license administrators will be able to understand the license terms without having to understand your implementation.

RLM is ***simple*** because we have kept the API to a minimum, and placed the license policy where it belongs – in the license file itself, where it is handled by RLM, not by your code.

Table of Contents

What's New in RLM Embedded v15.0

This section lists the new features along with pointers to the relevant sections in the manual. New license administration features are described in the License Administration manual.

What's new

- ***Prior to RLM v15.0, the linux ethernet hostid code would scan 5000 devices to get ethernet MAC addresses. In v15.0, the environment variable RLM_LINUX_ETHERNET_ITERATIONS will set the number of devices to check for MAC addresses.*** For more information, see A Note about Linux Ethernet hostids on page 155 for more information.

New License Keywords

- None

API additions

- None

API changes

- None

Activation changes

- ***The Activation pro debuglog now includes the PID of the license generator process,*** so that it is easier to interpret the log in the case where multiple instances of the license generator are running simultaneously. See “Debugging Tab” in the Activation Pro manual for more information.
- The “text to prepend to license” field has been increased from a maximum of 1024 to 30720 bytes. See “Product Definitions” in the Activation Pro manual for more information.

Table of Contents

Installing RLM Embedded

To install RLM Embedded, follow these steps:

● First, Download the kit from the Reprise website

To download RLM Embedded, go to the Reprise website download area, enter your username and password, and select the kit(s) you want to download. Save these on your system, then uncompress and (on unix) extract the binaries with the *tar xvf* command. [Note that there is not a separate RLM Embedded kit – RLM Embedded is controlled by a license from Reprise Software, and you use the standard RLM kit.]

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" (Unix) or ".exe" (Windows) appended:

Platform	Platform Name	Kit file name
HP-UX on PA-Risc	hp_h1	hp_h1.tar.gz
HP-UX 64-bit on PA-Risc	hp64_h1	hp64_h1.tar.gz
IBM AIX 32-bit	ibm_a1	ibm_a1.tar.gz
IBM AIX 64-bit	ibm64_a1	ibm64_a1.tar.gz
Linux on Intel X86	x86_l1, x86_l2	x86_l1.tar.gz, x86_l2.tar.gz
Linux 64-bit on Intel	x64_l1	x64_l1.tar.gz
Mac on Intel X86	x86_m1	x86_m1.tar.gz
Mac on PPC	ppc_m1	ppc_m1.tar.gz
Solaris 32-bit on Intel	x86_s1	x86_s1.tar.gz
Solaris 64-bit on Intel	x64_s1	x64_s1.tar.gz
Solaris on Sparc	sun_s1	sun_s1.tar.gz
Solaris 64-bit on Sparc	sun64_s1	sun64_s1.tar.gz
Windows 64-bit (visual C 2010-2013)	x64_w3	rlm.vX.YBLZ-x64_w3.exe
Windows 32-bit (visual C 2010-2013)	x86_w3	rlm.vX.YBLZ-x86_w3.exe
Windows 64-bit (visual C 2015 and later)	x64_w4	rlm.vX.YBLZ-x64_w4.exe
Windows 32-bit (visual C 2015 and later)	x86_w4	rlm.vX.YBLZ-x86_w4.exe
Java for Unix (requires x86_l2, x64_l1, or x86_m1 kit. x86_l2 only prior to RLM v6)	java_unix	java_unix.tar.gz

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.

● Next, unpack the kit and install

- For the majority of cases using a C-compiler, follow the instructions in this section.
- For information on using RLM with Java, see Using RLM Embedded with Languages other than C/C++ on page 60.
- For information on using RLM Embedded in a cross-development environment, see Building the RLM Embedded kit using a cross-compiler on page 17.

To unpack the kit and perform the installation, follow these steps:

At the shell prompt on Unix:

```
% gunzip platform.tar.gz
% tar xvf platform.tar
% ./INSTALL
%  # update src/license_to_run.h if required
%  # Your license for RLM comes via email from Reprise Software.
%  # RLM kits are pre-built with demo licenses valid for
%  # approximately two months from date of release.
% cd platform
% make
```

Note: RLM requires a license to operate from Reprise Software.

On Windows, the kit is in a Windows installer executable. Run the installer, whose name is rlm.v***ver-platform***.exe, where ***ver*** is the RLM version and ***platform*** is the RLM platform name. For example, rlm.v13.0BL1-x86_w4.exe is the installer for v13.0BL2 on the x86_w4 (Windows 32-bit, VS2015 and above) platform. The installer asks where you would like to install RLM - the default is in your *My Documents* folder. The installer will create the folder Reprise\rlm.***ver-platform*** (where ***ver*** and ***platform*** are as above) in your My Documents folder if you take the default.

You have the option in the Windows installer to specify that you would like the installer to copy your key pair from another RLM installation into the new installation area. This is useful if you are upgrading your RLM version or installing RLM on another system at the same release level, and wish to use the same key pair so as to have compatible license signatures across the installations. If you do not wish to specify the location of another key pair, leave the box blank.

RLM kits are pre-built for ISV "demo", with licenses that expire in 30-60 days after the release date. If your demo license has expired, you will need to put the new license you received from Reprise Software into the file src\license_to_run.h. If you have purchased RLM, you will need to edit src\license_to_run.h to replace the license there with your permanent license, and you will also need to edit src\rlm_isv_config.c and the makefile in the binary directory (x86_w* or x64_w*) to change your ISV name. If you plan to use the example license file example.lic in the platform directory, edit the file, and change all instances of "demo" to your ISV name.

Note: RLM requires a license to operate from Reprise Software.

You have 2 options for building 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.

To build using Visual Studio/Visual C++:

1. The platform directories (x86_w* and x64_w*) 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_w3, double-click on x86_w3.vcproj. In x64_w4, double-click on x64_w4vcproj, etc.
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.

You may be prompted to allow Visual C++ to convert the project to a 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
   - Create a command window and run a batch file provided by Microsoft to set up your command window for the next step. The batch file is Program Files [(x86)]\Microsoft Visual Studio <version>\VC\vcvarsall.bat
     -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

A note about OpenSSL

Note: as of v12.0 on Linux and v12.2 on Windows, RLM uses a private name space for the OpenSSL routines, so the need to remove those modules from the RLM library to avoid conflicts with other OpenSSL implementations that you link into your application has gone away, and you can ignore the remainder of this paragraph. If you are using an earlier version of RLM and wish to build a client library on Unix systems which does not contain any of the OpenSSL library routines, execute the **make rlm_nossl.a** command after installing your kit. The resulting library can be used to link your application if you use OpenSSL as part of your application and you use a different OpenSSL version.

Building the RLM Embedded kit using a cross-compiler

On certain platforms (e.g. arm_l1 and xpi_l1), the rlm kit must be cross-compiled on a host system which doesn't run the target instruction set. For these platforms, follow the directions here (Note: these directions are for Unix systems only, to do cross-development on Windows, you are on your own. See the makefile):

To unpack the kit and perform the installation, follow these steps:

At the shell prompt on Unix:

```
% gunzip platform.tar.gz
% tar xvf platform.tar
% ./INSTALL
%  # update src/license_to_run.h if required
%  # Your license for RLM comes via email from Reprise Software.
%  # RLM kits are pre-built with demo licenses valid for
%  # approximately two months from date of release.
% cd platform
```

At this point, on a “normal” RLM platform, you would simply type “make”. However, in a cross development environment, the make process is split into 4 or 5 steps. In these instructions, we will refer to the two systems as the ***host*** (the system with the cross-development tools), and the ***target*** - the target system which does not have development tools.

1. First, on the ***host*** system (the one with the cross-development tools):

   % make step1

2. Next, copy rlmgenkeys to the ***target***; run rlmgenkeys; copy rlm_privkey.c and rlm_pubkey.c back to the ***host*** system into the src directory.

3. Next, on the ***host*** system:

   % make step3

4. Next, copy the kit (the whole directory, e.g. arm_l1) to the ***target***

5. Next, on the **target**: (optional, only if you have a full client-server RLM kit).

   % make step5

Your kit is now built on the target and ready to use.

Note: skip steps 1 and 2 if you have a key pair from another rlm platform, and put the keys into the src directory on the ***host*** system; start from step 3 above. Skip step 5 if you have a client-only kit, or if you do not care about creating an ISV.set settings file.

RLM kit layout

Each RLM kit (for a particular platform) is contained in 3 or 5 subdirectories:

- Machine-independent subdirectory (src)
- Machine-independent examples subdirectory (examples)
- Machine-dependent subdirectory (name varies for each platform)

In addition, on Windows, there is an additional directory:

- a directory of .NET support files called "dotnet".

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)

RLM-Embedded is a subset of RLM, so you will use the full RLM kit.

Current RLM platform names are:

Platform	Directory Name	Notes
HP-UX on PA-Risc	hp_h1	
HP-UX 64-bit on PA-Risc	hp64_h1	

IBM AIX 32-bit	ibm_a1	
IBM AIX 64-bit	ibm64_a1	
Linux on ARM	arm_l1	Client-only kit
Linux on Intel X86	x86_l1, x86_l2	
Linux (64-bit) on Intel	x64_l1	
Linux on PPC	ppc_l1, ppc64_l1	
Linux on Xeon PI coprocessor	xpi_l1	Client-only kit
MAC on Intel X86	x86_m1	
MAC (64-bit) on X86	x64_m1	
MAC on PPC	ppc_m1	
Solaris (32-bit) on Intel	x86_s1	
Solaris (64-bit) on Intel	x64_s1	
Solaris on Sparc	sun_s1	
Solaris (64-bit) on Sparc	sun64_s1	
Windows 32-bit	x86_w3	Visual Studio 2010-2013
Windows 32-bit	x86_w4	Visual Studio 2015 and later
Windows 64-bit	x64_w3	Visual Studio 2010-2013
Windows 64-bit	x64_w4	Visual Studio 2015 and later

RLM Kit Contents

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 Embedded
RELEASE_NOTES	Release notes for this version of RLM
RLM_Reference.txt	Pointer to RLM documentation on website
VERSION	RLM kit version information (not on client-only kits)

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[tm].
example.opt	Example license administration option file
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 for using rehostable hostids and revoking them

rlm_transfer.c	Example ISV-defined server transfer code
rlmclient.c	Example rlm application program
roam_example.c	Example code to implement license roaming
unsuppoprted	Directory of unsupported example programs (fortran interface, python interface)

Each Unix 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	UNUSED

	utility	
x86_w*.vcproj, x64_w*.vcproj	Visual Studio/Visual C++ project for configuring the kit	

The Java directory (java_unix, java_win) is unused in RLM Embedded.

The dotnet directory (RLM .NET support – Windows only) contains:

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

Table of Contents

Integrating RLM Embedded Into Your Product

OVERVIEW - Software License Management Basics

If you have used other license management products, you can skip this section. If this is your first time, however, we have included an overview of how license management products operate.

RLM Embedded is similar in structure to most popular license managers. RLM Embedded consists of 3 major components:

1. a client library
2. license utilities
3. a text file which describes the licenses granted (the *license file*).

Your application is linked with the client library which provides access to the license management functions.

The RLM client library (linked into your application) is controlled by license authorizations stored in a text file called the *license file*.

Most license managers provide APIs with calls to control many of the aspects of licensing behavior, as well as options within the license servers to control licensing behavior. 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 all these options to the license file, where they are visible and understandable by everyone. In general, 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. The API is simpler, and more standard from ISV to ISV. This prevents license management confusion in license administrators. We learned this the hard way when we supported hundreds of customers in the past, and applied these lessons to the design of RLM.

INTEGRATING RLM Embedded Into Your Product - The 6 Steps

In order to add license management capabilities to your product, there are 6 main steps:

1. Decide on your Licensing Strategy
2. Create your Keys (public/private key pair)
3. Add RLM Embedded API calls to your application
4. Configure and build your RLM Kit
5. Package your software for shipment
6. Create licenses for your customers

These steps are described in the following sections.

1. Decide on your Licensing Strategy

RLM Embedded 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 Embedded)
- Product name
- Highest Version supported
- the node identification for the license
- Expiration date

Before you integrate RLM Embedded 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.

2. Create your Keys (public/private key pair)

Before you use RLM Embedded, you need to create a *public-private key pair*. **You should only do this one time**, since 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.

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 and your ISV server
- 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 Embedded, you will be asked for the location of these two files, so that the new version will generate compatible keys with your older versions.

3. Add RLM Embedded API calls to your application

Everything you need for most applications is contained in the 8 functions in the RLM Embedded core API.

These functions are described in Appendix A – you can follow the links in the following table:

rlm_init() - initalize licensing operations with RLM.
rlm_close() - Terminate licensing operations with RLM.
rlm_checkout() - Request a license.
rlm_checkin() - Release a license.
rlm_errstring() - Format RLM status into a string.
rlm_stat()- Retrieve RLM_HANDLE status.
rlm_license_stat() - Retrieve RLM_LICENSE status.
rlm_get_attr_health() - Check license status.

If you have special licensing needs that are not addressed by these functions, see Appendix A – RLM Embedded API on page 81 which lists all RLM Embedded API functions.

4. Configure and build your RLM Embedded Kit

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

- Install your RLM Embedded license.
- Create your public/private key pair, which is done one time only and which was done in step #2, above. (See Create your Keys on page 23).
- Configure your RLM Embedded parameters.
- Modify the makefile to change the ISV name "demo" to your ISV name (if you previously installed a demo kit). Note: you can skip this last step if you have an evaluation kit.

To install your RLM Embedded 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).

An example *license_to_run.h* file is shown here (this is a demo license which expired on 1-jul-2007):

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

            COPYRIGHT (c) 2007-2011 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:    License to use RLM
 *
 *      Replace the RLM license on the four lines after:
 *
 *              #define RLM_LICENSE_TO_RUN       \
 *
 *      with the license you received from Reprise Software.
 *
 *
 */

#ifdef RLM_LICENSE_TO_RUN
#undef RLM_LICENSE_TO_RUN
#endif

#define RLM_LICENSE_TO_RUN \
  "1-jul-2007 \
  sig=\"c2N250Z4hGt2HCMWNcye*Xe35YI8LGZf0ihLbEfJ8Bfe~zS0IFwu7R78Iye1ao\""


#define RLM_ISV_NAME "demo"
```

Your applications and your ISV license server 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 on page 23).
- A file of RLM Embedded 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.

Customizing RLM Embedded with rlm_isv_config

rlm_isv_config.c contains calls to:

- set up your ISV name
- install your RLM Embedded license (do not change this call)
- disable the RLM Embedded clock windback detection for expiring licenses
- enable or disable Windows disk serial numbers which require admin access to use
- register ISV-defined hostids
- include or exclude code for optional hostids (e.g., dongles, etc)
- specify the types of hostids which Activation Pro will accept
- specify the URL of your activation server (for Alternate Server Hostids)

Edit this file before compiling your license generator or applications.

NOTE: your ISV name is case-insensitive.

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

An example rlm_isv_config.c file is shown here. Note that there are many options which pertain to license servers – you can ignore all of these for RLM Embedded:

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

            COPYRIGHT (c) 2005, 2011 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.

*****************************************************************************

            COPYRIGHT (c) 2005, 2014 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:     rlm_isv_config.c - configuration data for ISV
 *
 *       M. Christiano
 *       11/25/05
 *
 */

#include "license.h"
#include "license_to_run.h"

/*
 *      Define "INCLUDE_RLMID1" to include support for RLMID1 dongles.
 *      Comment out to remove aladdin dongle support.
 *
 *      Note: The RLMID1 dongle code is always included in
 *      your license server.  This setting is only for your applications, and
 *      only needs to be set if you are issuing licenses that are nodelocked
 *      to a dongle.
 *
 *      Including the RLMID1 dongle code increases the size of
```

```
 *      your applications by approx 900Kb on 32-bit windows, plus involves
 *      a small delay at application startup time, even if you are not using
 *      a dongle.
 *
 *      If you are not planning to issue licenses which are node-locked to
 *      rlmid devices, Reprise Software recommends leaving these options turned
 *      off (ie, leave the "#if 0" on the next line).
 */

#if 0
#define INCLUDE_RLMID1
#endif

#ifdef INCLUDE_RLMID1
extern void _rlm_gethostid_type1(RLM_HANDLE, L_HOSTID);
#endif


void
rlm_isv_config(RLM_HANDLE handle)
{

/*
 *      Set ISV name
 *
 *      NOTE: IF you are evaluating RLM, DO NOT change the ISV
 *            name, or your license keys will no longer work.
 *            For eval kits, the name on the next line MUST
 *            be "demo".
 *
 *      NOTE: Your ISV name is, in general, case-insensitive.
 *            The ONLY exception to this is when it is used as
 *            a lockfile name using a FLEXlm-compatible lockfile.
 *            In this case (and this case only), the case of the
 *            name you enter here is important.  Note that even in
 *            this case, ONLY THE LOCKFILE NAME uses the exact case
 *            you enter - every other place in RLM uses a lowercase
 *            version of this name.
 *
 *      Beginning in RLM v7.0, your ISV name is contained in
 *      "license_to_run.h".  If you need to alter the case of the
 *      name for a compatible FLEXlm lockfile, you should do it there
 *      and leave the next line as it is.
 *
 */
        rlm_isv_cfg_set_name(handle, RLM_ISV_NAME);


/*
 *      Set RLM license - do not modify this line
 */
        rlm_isv_cfg_set_license(handle, RLM_LICENSE_TO_RUN);


/*
 *      Set oldest allowed server version.
 *
 *      The next setting controls the oldest RLM license server
 *      version with which your application will work.
 *
 *      The 3 parameters are rlm version, revision, and build (in
 *      that order).
 *
 *      If you leave this set to 0, 0, 0, your application will
 *      attempt to work with the oldest available RLM server.
 *
 *      You should only set this if you are concerned with an older
 *      server in the field which has been hacked, otherwise, you should
 *      leave it set to 0, 0, 0.
 *
 *      (Note: Do not set this to anything between 0,0,0, and
 *       9,0,0).  Servers older than v9.0 will appear to be v0.0)
 *
 */
        rlm_isv_cfg_set_oldest_server(handle, 0, 0, 0);

/*
```

```
 *      Set ISV server settings file compatibility
 *
 *      The next setting controls what versions of RLM your
 *      ISV server settings file will work with.   You can enable
 *      it for all earlier versions (> v6), or later versions or both.
 *      The 2nd parameter enables earlier versions if non-zero, the
 *      3rd parameter enables later versions if non-zero.  Note that
 *      "earlier" and "later" are relative to the version of your
 *      settings file.  So, if you create the settings file with RLM v8,
 *      "earlier" means v6 and v7, while "later" means v9 and above.
 *
 *      default is: rlm_isv_cfg_set_compat(handle, 0, 1); - sets compatibility
 *                      with later versions, but not earlier ones.
 */
       rlm_isv_cfg_set_compat(handle, 0, 1);

/*
 *      Setup virtual machine enable/disable.
 *
 *      By default (if you do not modify the following call), RLM
 *      will refuse to run a license server on a virtual machine.
 *      If you want license servers to run on virtual machines, set the 2nd
 *      parameter of the next call to a non-zero value.
 *
 */
       rlm_isv_cfg_set_enable_vm(handle, 0);

/*
 *      Beginning in RLM v10.0, roaming is disabled for servers that
 *      use transient hostids (ie, dongles, or ISV-defined transient hostids).
 *      If you want to enable roaming on these servers, set the 2nd
 *      parameter of the next call to 1.
 */
       rlm_isv_cfg_set_enable_roam_transient(handle, 0);

/*
 *      Beginning in RLM v10.0,  you have the option of turning ROAMED
 *      licenses into "single" licenses.  Prior to RLM v10.0, all ROAMED
 *      licenses were nodelocked, uncounted.
 *      If you want your roamed licenses to be "single" licenses, set the
 *      second parameter of the next call to 1.
 */
       rlm_isv_cfg_set_roam_single(handle, 0);

/*
 *      Beginning in RLM v10.0 it is possible to disable the clock windback
 *      check.  In previous versions it was always enabled.  Passing a 1 in
 *      the second argument of the following function call disables the
 *      windback check; passing 0 leaves it enabled (the default).
 */
       rlm_isv_cfg_disable_clock_windback_check(handle, 0);

/*
 *      FLEXlm(R)-style lockfile compatibility.
 *
 *      Set to non-zero to use a FLEXlm-style lockfile.  For windows
 *      systems, a value of 1 uses the 'C' drive always, whereas a
 *      value > 1 will use the system drive.  FLEXlm (up to version
 *      10.3, at least) puts the lockfile on the 'C' drive.
 *
 *      Reprise Software recommends setting this to 1 if you want to
 *      use FLEXlm-compatible lockfiles.
 */
       rlm_isv_cfg_set_use_flexlm_lockfile(handle, 0);

/*
 *      The Windows disk serial number hostid code can return hostids
 *      that are usable only by processes running with admin rights if
 *      running with admin privileges.  If an application is installed
 *      and a license activated by an admin user, it's possible that
 *      a non-admin user will not be able to use the application because
 *      it can't read the disk serial number.  Beginning in RLM v10.0,
 *      you can disable the use of disk serial number hostids that are
 *      usable by admins only.  If you want to do so, change the second
 *      parameter of the next function to 0.
 */
#ifdef _WIN32
```

```
        rlm_isv_cfg_set_use_admin_disksns(handle, 1);
#endif

/*
 *      Beginning in RLM v10.0,  RLM's license transfer capability also
 *      allows for disconnected operation on the destination server.
 *      This capability only requires that an "rlm_roam" license be
 *      present on the destination server.  You can ship an rlm_roam
 *      license to your customer and have them install it on every
 *      destination server, or you can simply put it into the next
 *      call, in which case, no separate license file will be needed
 *      on the destination license server.
 *
 *      To enable this, set the 2nd parameter of the next call to a valid,
 *      signed rlm_roam license (enclosed in "<>") in place of the
 *      last argument.  This license should be a static string
 *      which is available for the lifetime of the server.
 *
 *      This license MUST have the following parameters:
 *              version: "1.0"
 *              exp: "permanent"
 *              count: "uncounted"
 *              hostid: "any"
 *              NO other parameters
 *
 *      for example:
 *
 *      rlm_isv_cfg_set_server_roam(handle, "<LICENSE rlm_roam your-isvname 1.0
uncounted hostid=any sig=xxxxxxx>");
 */
        rlm_isv_cfg_set_server_roam(handle, (char *) 0);

/*
 *      Beginning in RLM v10.0,  RLM can broadcast to find a license
 *      server as a last resort, if all the normal methods to find
 *      the server fail.  This capability is enabled by default.
 *
 *      To disable this, set the 2nd parameter of the next call to 1.
 */
        rlm_isv_cfg_disable_broadcast(handle, 0);

/*
 *      Beginning in RLM v11.0, the client can specify that
 *      it will not use a generic license server.
 *      If you want to disable generic servers, set the 2nd
 *      parameter of the next call to 1.
 *      If you disable generic servers and your application
 *      attempts to connect to a generic server, it will
 *      receive an RLM_EH_SERVER_REJECT error upon connection.
 */
        rlm_isv_cfg_disable_generic_server(handle, 0);

/*
 *      Beginning in RLM v10.1, licenses can be cached on the client
 *      side with the use of the "client_cache" license attribute.
 *      This capability must be enabled with the following call.
 *      If the 2nd parameter is 1, client caching is enabled, if 0,
 *      caching is disabled.
 *      Note: this call has no effect on HP systems.
 */
        rlm_isv_cfg_enable_client_cache(handle, 1);

/*
 *      Beginning in RLM v10.1, license servers can return one
 *      valid license to the application which is then verified on
 *      the client side.  This check helps ensure that the license
 *      server hasn't been modified.  To enable this checking set
 *      the second parameter of the next call to 1.  If you enable
 *      this, please read the section titled "Server Integrity Checking"
 *      in the "Securing Your Application" section of the Reference
 *      Manual so that you understand the errors which can be generated
 *      as a result of this call and how you should proceed.
 */
        rlm_isv_cfg_enable_check_license(handle, 0);

/*
 *      Beginning in RLM v11.0, you can specify which types of
```

```
 *      hostids that Activation Pro will accept from an activation
 *      request.   Prior to v11.0, the only 6 types of acceptable
 *      hostids were: rehostable, isv-defined, rlmid, ethernet,
 *      disk serial numbers and native 32-bit hostids.
 *      In the following call, you can set the default hostids that
 *      your Actpro server will accept.  To get the pre-v11 behavior,
 *      set the 2nd parameter as shown.  Hostid type definitions in license.h
 *
 */

#if 0
{
  int allowed_types =   RLM_ACTPRO_ALLOW_REHOST | RLM_ACTPRO_ALLOW_ISV |
                        RLM_ACTPRO_ALLOW_RLMID | RLM_ACTPRO_ALLOW_ETHER |
                        RLM_ACTPRO_ALLOW_DISKSN | RLM_ACTPRO_ALLOW_32 |
                        RLM_ACTPRO_ALLOW_ASH;

        rlm_isv_cfg_actpro_allowed_hostids(handle, allowed_types);
}
#endif

/*
 *      Beginning in RLM v11.2, license servers can utilize
 *      Alternate Server Hostids.  These hostids are activated
 *      from Activation Pro by the ISV server, which needs to
 *      know the URL of the activation server.
 *      If you use Reprise's hosted activation service, the default
 *      (hostedactivation.com) is correct.  For all others, set your
 *      activation server url here.  Note that this URL pointer must
 *      remain valid as long as the RLM_HANDLE is in use.
 */
       /*** rlm_isv_cfg_set_url(handle, "hostedactivation.com"); ***/

/*
 *       If you want to add ISV-defined hostids to the ISV server,
 *       use code similar to the following for each new hostid type
 *       you would like to add.
 */
#if 0
         stat = rlm_add_isv_hostid
                  (
                           handle,  /* RLM_HANDLE passed in */
                           "keyword here",  /* Hostid keyword you chose */
                           YOU_DEFINE_HOSTID_TYPE, /* Your hostid type (int)
                                                      > RLM_ISV_HID_TYPE_MIN */
                           transient,       /* (int) == 0 if hostid does not
change.
                                               Non-zero if it does change, e.g., if
                                               your hostid is a dongle, it can
                                               change if someone unplugs it, so
                                               you should set transient non-zero */
                           get_type_hostid  /* Your function to determine the
                                               hostid value */
                  );
         if (stat)
         {
                  printf("ERROR: add hostid type returns %d\n", stat);
         }
#endif


/*
 *      To include RLMID1 dongle code, be sure INCLUDE_RLMID1 is defined above.
 */

#ifdef INCLUDE_RLMID1
        rlm_isv_cfg_set_use_hostid(handle, RLM_HOSTID_RLMID1,
                                                _rlm_gethostid_type1);
#endif
}
```

5. Package your software for shipment

With RLM Embedded, 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 Embedded 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 Embedded License Administration Tools (rlmutil, rlmhostid, rlmreread, rlmswitch, etc) in your distribution kit.
- If you use the optional rlmID1 hardware keys with your product, make sure you ship the Aladdin utilities with your distribution kit. See Appendix F – Optional Hostid Installation Instructions on page 159 for more details.
- Review the Best Practices for RLM Embedded Integration section and ensure that your product and installation are well-behaved.

6. Create licenses for your customers

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.

Format of the license file

The license file consists of lines of readable text which describe the license server node, some parameters of the license server binaries, and the actual license grants to your customers. For a complete description of the license file format, see *The License File* on page 39.

Types of Licenses

While there is a single format for the license file, the licenses you create can have many different meanings. For more details, see *Creating Licenses* on page 49.

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 other license parameters. For more information on rlmsign, see *Creating Licenses* on page 49.

License creation API

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, *rlm_sign_license(),* is available for cases where this is not practical. For details on the usage of *rlm_sign_license(),* see *Appendix A - RLM Embedded API* on page 81.

License creation GUI

In addition to rlmsign and rlm_sign_license(), RLM Embedded provides a GUI for license generation, rlmgen. The rlmgen program is described in *Creating Licenses – rlmgen* on page 52.

Internet Activation

RLM Activation Pro allows the ISV to give a customer an *activation key* which then allows the customer to retrieve their license from the ISV 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 are in the RLM Activation Pro manual. RLM Activation Pro is an optional product.

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. Currently, there is one Reprise Product Name in use that applies to RLM Embedded, however do not use any license name starting with “rlm_”:

- *rlm_demo* - This product name is used by RLM to enable ***Detached Demo***^{TM} licenses for your products.

Note also that license ***replace*** processing uses the single-character product name '*' to indicate all licenses, so you should avoid a product name of “*”.

The first 5 steps are done once or perhaps once per release of your software. The final step is done each time you sell your software to a customer. You might also want to take a look at the RLM Example Client Program, in appendix C.

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 **x86_w3, x86_w4, x64_w3, or x64_w4.**

• 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
- wbemuuid.lib
- commsupp.lib
- ole32.lib
- oleaut32.lib
- libcmt.lib

In addition, include these libraries if you're using VC++ 2015 or later:

- libvcruntime.lib
- libucrt.lib

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

Using optional RLMID hostids

RLM supports a number of optional hostid choices. These are generally called *rlmid*N, e.g. *rlmid1*. Each of these optional hostids has individual requirements when you build your software and when you ship it.

RLM Embedded currently supports one optional hostid choice:

- rlmid1 - a hardware key manufactured by Aladdin Knowledge Systems (now SafeNet, Inc.)

Each optional hostid has specific requirements both when you build your product and when you ship it. In addition, each optional hostid is available only on certain platforms.

In your application, you need to enable the various *rlmid* hostids should you chose to use the particular hostid for nodelocked licenses. You do this in rlm_isv_config.c

Platform Support

The following table lists the first RLM version in which support is available for the particular *rlmid* device. Only the listed platforms are supported.

platform	rlmid1
x86_l2	v11.1
x64_l1	v11.1

x86_w1	v5
x86_w2	v5
x86_w3	v9
x64_w2	v5
x64_w3	v9

Startup Delay

Adding support for rlmid devices will cause your application to experience a short delay at startup time. We tested the following scenarios, using a loop of 2000 iterations which does *rlm_init(), rlm_checkout(), rlm_checkin(), and rlm_close()* of an uncounted license, on a 3GHz AMD desktop system:

Scenario	Total time	loops/second	seconds/loop
no rlmid support, license locked to ANY	30 seconds	66.67	0.015
rlmid1+ rlmid2 support, license locked to ANY	37 seconds	55.55	0.018
rlmid1+ rlmid2 support, license locked to rlmid1	40 seconds	50	0.02

rlmid1

The rlmid1 devices are manufactured by Gemalto (formerly SafeNet (formerly Aladdin Knowledge Systems)). They are a small purple USB hardware key containing an internal serial number. The hostid is printed on the outside of the key as "rlmid1=xxxxxxxx" where *xxxxxxxx* is the serial number (hostid) of the key.

Enabling rlmid1 devices in your application

Windows kits contain everything required to use the optional rlmid hostids, Client-side support is not included by default, and you must follow these instructions to add support for using rlmid1 devices in your application.

Proceed as follows:

In order to enable rlmid1 devices for nodelocking in your application, locate the following 3 lines in rlm_isv_config.c:

```
#if 0
#define INCLUDE_RLMID1
#endif
```

and change the first line from "#if 0" to "#if 1". Re-build your rlm client library (by typing "make" on Unix, or "nmake" on Windows).

When you re-link your application, you will need to include the library *rlmid1.lib* (on Windows)

or rlmid1.a (on Linux) from the RLM binary directory.

At this point, your application is enabled to use rlmid1 devices. You will notice that your application grows by approx 900kb on Windows, and there will be a short delay the first time you request a license, when RLM Embedded attempts to determine the ID of any rlmid1 devices connected to the local system.

For information on shipping your product with *rlmid1* devices, see Appendix F – Optional Hostid Installation Instructions on page 159.

Advanced API Functions

There are some options you can set within your application. Generally, the defaults will work, but if you want more control, you can look at Appendix A for a description of all the available RLM Embedded API functions.

Clock Tampering Detection

RLM Embedded will attempt to check for system clocks that have been set back when it checks out a license that expires. This check will happen in the license server for floating licenses or in the client for node-locked licenses. This check is automatic; you do not need to modify your application in any way to effect this check.

RLM Embedded hostids

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

RLM Embedded'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	32-bit hostid, native on Unix, non X86 based platforms	hostid=10ac0307	Volume serial number on windows, not recommended
disksn (See note below)	Hard disk hardware serial number	hostid=disksn=WD-WX60AC946860	Windows only
ip (or internet)	TCP/IP address	hostid=ip=192.156.1.*	always printed as "ip="
ether	Ethernet MAC address	hostid=ether=00801935f2b5	always printed without leading "ether="
user	User name	hostid=USER=joe	
host	Host name	hostid=host=melody	

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 32
```
or
```
rlmutil rlmhostid internet
```

A Note about Windows disksn hostids

Some disk serial numbers on Windows are only accessible to a process running with admin privileges. To disable use of disk serial numbers that only admins can use, see the call to *rlm_isv_cfg_set_use_admin_disksns()* in rlm_isv_config.c.

Table of Contents

Best Practices for RLM Embedded Integration

Our experience supporting thousands of 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 Embedded. Adherence to these guidelines, while not strictly mandatory, will be greatly appreciated by your 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_akey(), rlm_license_count() and 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 Embedded 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 Embedded Into Your Product, and The License File, there are a few ways that your application and license server can locate the licenses they need to operate:

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

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 license administrators 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.
- ***Include a folder for licenses in your installed product folder tree.***

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 license administrators. In general, the more fragmented into separate license domains an application becomes, the less license administrators understand the licensing behavior and the less satisfied they are. In an ideal world (from the license administrator'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.

Table of Contents

The License File

The license file contains information which describes all the licenses granted from the ISV to your customer. RLM Embedded has only one basic license type: ***node-locked, uncounted.*** The various attributes modify this basic license type.

License Files have 3 types of lines:

1. LICENSE Lines which describe license grants from the ISV to your customer (FEATURE is an alias for LICENSE)
2. UPGRADE lines which upgrade the version number of some or all LICENSEs, and
3. Comment lines

Applications and License Administration Tools locate the license file using The License Environment.

Comments in license files

Lines beginning with '#' are treated as comments and not interpreted by RLM. Comments may be added to a license file without invalidating the signatures of licenses, but should not be added between the lines of a multiple-line license. Here is an example:

```
#
# Licenses served by host gt2
#
HOST gt2 0000a74f88ce 5053
ISV reprise
#
# Original license for v3.0 (10 seats)
#
LICENSE reprise joe 3.0 permanent 10 _ck=f81efcf79a
sig="60PG451KTXVQ0WYBX785XAKTDKUCHB7T683Y2MG22M088S8UAFR0VKPMFGPKH
 4XW4H5QQ8JSFFJG"
#
# v4.0 license (5 additional seats)
#
LICENSE reprise joe 4.0 permanent 5 _ck=3b1efcd48c
sig="60P045145JSKEJSR48V3GXCX29S8TM5TKE91TS022HW0XAEEWH82DRTCJB830AW
 EV62MUE2N7C"
```

Note that prior to RLM Embedded v9.3, the comment character was not strictly required on comment lines. With improved error checking in 9.3 however, the comment character is required.

Special License Names

Any product name beginning with "rlm_" is reserved to Reprise Software.

Legal characters in the license file

In general, all license file fields are white-space delimited, meaning that no data item can contain embedded spaces, tabs, newlines or carriage returns. In addition, the following six characters are illegal in data items in the license (and options) file: "<", ">", "&", single quote ('), back-quote (`) and double-quote ("). ISV license names cannot begin with the characters "rlm_".

Note that all lines in license files ***must*** be shorter than 1024 characters. Anything over 1024 characters will be truncated.

Everything in the license file is case-insensitive, with the exception of short (~62-character) license keys (keys with bits/character of 6 - see Creating Licenses).

Note: any time RLM Embedded processes a *username*, it will replace any white space in the name with the underscore '_' character. This is true for *usernames* used as hostids. Also note that *usernames* are case-insensitive.

Order of lines in the license file

In general, the order of lines in the license file does not matter, with the following exception:

- LICENSE/FEATURE lines are processed in the order they appear in the license file. This means that you can bias the selection of licenses by the order they appear in the license file. For example, if you have licenses for product ABC versions 1.0 and 2.0, and your software requests version 1.0, the license you receive will depend on the order: if the 2.0 license appears first in the license file, and it is available, your application will receive a v2.0 license. If the v1.0 license appears first and it is available, you will receive a v1.0 license.

LICENSE Line

Format:

LICENSE *isv product version exp-date count* [sig=]*license-key* [optional parameters]

The LICENSE line defines the usage rights to a *product*. All fields in the license line are case-insensitive (with the exception of short, ie, less than 62-character, license keys), and none may be modified by the license administrator, with the exception of the parameters whose names begin with the underscore ("_") character.

Note: Prior to RLM v9.3, the license file parser did not reject optional keywords which were unknown – rather, it silently ignored them. Beginning in RLM v9.3, the parser will reject unknown keywords, so that more errors can be detected at license generation time rather than later. This means that some license templates which worked correctly pre v9.3 will no longer work. A couple of examples of this are:

LICENSE isv product v1.0 1-jan-2014 uncounted hostid=any key

(in this case, "key" is interpreted as an optional parameter and it is rejected. To fix this, change "key" to "sig").

Another example is a license in a string passed to rlm_init(). This license previously worked:

<valid license, without terminating '>' : <second license>

This will no longer work, as the parser interprets the ':' path separator character as an options field, which it rejects. To fix this, insert the trailing '>' character after the first license.

Fixed (positional) parameters

The first 6 parameters are required on every license, and are present in the order shown above.

- *isv* is the name of the ISV granting the rights.
- *product* is the name of the product for which license rights are being granted.
- *version* is the highest-numbered product version supported by this license, in the form "N.M". For example, 1.0, 2.37, or 2006.12 Each RLM Embedded license has a version number, of the form "*major.minor*". The version in the *rlm_checkout()* call must be less than or equal to the version in the license for the checkout to succeed. (Note: This comparison is done in the "normal" way, ie, 1.2 is greater than 1.10).

  The version can be used in a number of ways:

  You could make all your software ask for version 1.0 with all your licenses issued for version 1.0, and the version would never be an issue, unless and until you wanted to obsolete all the old licenses on a new release.

  You could put your product's version number in the rlm_checkout() call, then licenses for an older version of your product will not work with a newer version of the product.

  You can use a date-based version. To do this, you might put the year and month of release into the *rlm_checkout()* call in your application, then when you issue licenses, issue them either for this year and month when your customer's maintenance period ends.

  This allows your customer to use products released on or before the date in the license. Bear in mind that you would need to use the leading 0 in the month, since 2006.2 is greater than 2006.11, which might not be what you intend.
- *exp-date* is the date the license expires, in the form dd-mmm-yyyy, for example, 1-jul-2007. All licenses have a *expiration date*. If you prefer for your licenses to not expire, you can use the special expiration date of ***permanent***, which never expires (any date with a year of 0 is also non-expiring, e.g. 1-jan-0).

  Note: As of v9.4, RLM Embedded uses two checks to determine if the system clock has been set back. The first is a proprietary algorithm which does not access any other computers, and has been used in RLM since version 1.0. It is fast but sometimes returns false positives. If this check indicates clock windback, the second check is invoked. The second check involves connecting to a random NIST (National Institute of Standards and Technology) time server. If the second check succeeds, that is, the system time is not behind the time server time, RLM Embedded doesn't generate a clock windback error.
- *count* is the number of licenses granted. ***The count field defines the license type.*** See the License Models chapter on page 9 for a discussion of license types and modifiers. The ***license type*** is one of:
  - 0 or “uncounted” indicates an **uncounted license.**
  - **“**single” means a node-locked, single-use license. single is a special case of a **counted license**, but it is different from “1”. A license with a count of 1 is a regular counted license, and requires a license server. A license with the keyword “single” is a single-use, nodelocked license. This license does not require a license server, and in fact license servers will not process this license. single licenses are a convenient way to issue single-use licenses without the license administrator having to configure a license server.
- *license-key* is a digital signature of all the license data, along with the hostid on the HOST line, if present. If a license has a non-zero count, it always requires a HOST line. An uncounted license does not require a HOST line, and even if there is a HOST line, the hostid

of the license server is not used in computation of its *license-key*. The *license-key* will have "sig=" prepended after the license has been signed by the *rlmsign* utility.

Note that if the *license-key* is preceded by *sig*=, it can be present after any or all of the optional parameters.

In addition to the standard license attributes above, licenses can have the following optional license modifier attributes:

- **Locking: Node-locked** (uncounted or single)

RLM Embedded can lock a license in a variety of ways:

❑ A license can be *node-locked*. A node-locked license can only be used on a single node, as specified by the *hostid* of the license. For a description of the available hostids in RLM Embedded, see RLM Embedded hostids in the *Integrating RLM Embedded into your Product* chapter. The hostid in a license can be a *hostid list*, which means that the license is usable on *any* of the specified hostids.

❑ A node-locked license can be either *uncounted,* or *"single".* If it is *uncounted* or *single*, then the software only need verify that it is executing on the correct computer.

❑ To create a node-locked license, add the keyword **hostid=..** at the end of the license line. See the description of the LICENSE Line for more information.

❑ A license can be locked to a user. This is a special case of a *node-locked* license, and is accomplished using the hostid **user=...**. Note that any white space in a username is converted to the underscore ('_') character. Also note that usernames are case-insensitive.

- ***hostid=hostid-string (used for license locking)***

The optional hostid at the end of the line specifies that the licenses can only be used on the specified host. Uncounted licenses always require a hostid. Counted licenses generally do not have a hostid, but it could be present, in which case we would call this license a "node-locked, counted" license. (For a description of the various hostids that RLM Embedded supports, see Appendix E – RLM Hostids, on page 154.

The hostid on a LICENSE line can be a *hostid list*. The *hostid list* is a space-separated list of valid hostids, enclosed in double-quotes. The license can be used on *any* of the hostids in the list. The list can contain at most 25 hostids, and can be no longer than 200 characters.

For example, this hostid list would allow the license to be used in any of the 4 specified environments:

hostid="ip=172.16.7.200 12345678 rlmid1=83561095 user=joe"

- **Activation Key used to create this license**

akey=activation-key

When requested in RLM Activation Pro, the license generator will include the akey= keyword with the activation key used to fulfill the license. *akey=* first appeared in RLM v11.0.

- **Disable Computing Environment**

disable="computing-environment-list"

disable= specifies that clients running in the appropriate computing environment cannot use this license.

computing-environment-list is a list of different computing environment descriptions; if the application is running in any of these environments, the license will not be usable.

computing-environment-list is a space-separated list of the following environments (Note: put the list in quotes if more than one item is specified):

- **TerminalServer** - disable use on Windows Terminal Server and Remote Desktop.
- **TerminalServerAllowRD** – disable use on Windows Terminal Server but allow use via Remote Desktop
- **VM** - disable use on Virtual Machines.

Disabling TerminalServer is most useful for node-locked, uncounted licenses, if you do not want to allow multiple network users running remote sessions to make use of a single license. Note that you can't disable both TerminalServer and TerminalServerAllowRD – they are mutually exclusive.

Disabling Virtual Machines is useful for node-locked, uncounted licenses in order to prevent these licenses from being used on multiple virtual machines with the same hostid.

Example:
disable=TerminalServer

- **Expiration time**

exptime=hh:mm

Beginning in RLM v14.1, a license can be set to expire at a particular time on the expiration day. If the license includes an "exptime" keyword, the license will expire at this time on the expiration date rather than at midnight. If the exptime keyword isn't present, the license expires at midnight, as it has always done in previous versions of RLM. The expiration time, like the expiration date, is in local time.

- **License ID**

Any License Administrator can add ***_id=nnn*** to a license. "nnn" is a positive integer, less than 2**31, which is used to identify the license. If no ***_id=*** keyword is present, the id of the license is 0. The id of a license can affect license pooling as follows:

> A license that doesn't specify an id (or specifies 0), will pool with any other license that it would normally pool with. However, a non-zero id will only pool with the same same ID# (assuming all the other attributes make it eligible to pool).

Other than license pooling, the id can be used to select which licenses to apply an option (such as RESERVE). The id is not used in the computation of the license signature, and as such can be added or changed by the License Administrator.

- **License Issue Date**

If *issued=dd-mmm-yyyy* is specified in the license, this license issue date will be used in the computation of license replacement. If no issue date is present, the license start date is used. If neither is present, then this license will be replaced by any license specifying a *replace=* keyword with this license's product name.

- **License Options**

options = options_list

The *options* specification is used to encode options for the product license. The options field is a string (up to 64 characters in length) which is completely defined by the ISV. The options are used to calculate the license signature, but otherwise are unused by RLM. You can retrieve the options from a license with either the *rlm_product_options()* or the *rlm_license_options()* call. Note that if the string contains embedded white space, it must be enclosed within double quotes.

- **Platform Restrictions**

platforms=platform_list

RLM allows you to specify one or more platforms on which the application must be running. If a *platforms=platform-list* specification is contained in the license, the computer on which the application is running must be one of the specified platforms.

To specify one or more platforms, create a list of platform names. The *platform-list* consists of a list of RLM-defined platform names, which consist of a machine architecture and an operating system version/revision. Specify *platforms=* as a space-separated list of platform names with the trailing OS revision removed, as shown in the following table. Note that if you specify more than one platform, enclose the entire string in double quotes, e.g., platforms="sun_s x86_w sun64_s". Also note that while you can include the trailing revision number, it will not be used by RLM in any comparisons, so including it may lead to confusion.

Platform	RLM Platform name	string to use in *platforms=*
HP-UX on PA-Risc	hp_h1	hp_h
HP-UX 64-bit on PA-Risc	hp64_h1	hp64_h
IBM AIX 32-bit	ibm_a1	ibm_a
IBM AIX 64-bit	ibm64_a1	ibm64_a
Linux on Intel X64	x86_l1, x86_l2	x86_l
Linux 64-bit on Intel	x64_l1	x64_l
MAC on Intel X86	x86_m1	x86_m
MAAC on Intel (64-bit)	x64_m1	x64_m
MAC on PPC	ppc_m1	ppc_m
Solaris 32-bit on Intel	x86_s1	x86_s

Solaris on Sparc	sun_s1	sun_s
Solaris 64-bit on Sparc	sun64_s1	sun64_s
Windows on Intel x86	x86_w3, x86_w4	x86_w
Windows 64-bit on Intel	x64_w4, x64_w4	x64_w

- **Replacement Licenses**

  replace[=product_list]

  In order to render ineffective one or more licenses which you have already issued, use the *replace[=product-list]* option in the new license. *replace=* causes RLM to ignore the "replaced" license(s). Beginning in RLM v11.0, if *product_list* is the single character '*', **all** licenses will be replaced.

  Note: If you specify *replace*, you must also specify either *start=* or *issued=*.

  ***replace* operates as follows:**

  - licenses from the *product_list* will be replaced (all licenses if *product_list* is '*'). If *product-list* is not specified, then the product name of the license containing the replace keyword will be the only product to be replaced.
  - if the license with the *replace* keyword specifies an *issued=* date, then this is the "*replacement date*".
  - if the license with the *replace* keyword does not have an *issued* date, then the "*replacement date*" is the *start* date of the license.
  - if the license contains neither an *issued* date nor a *start* date, no licenses will be replaced.
  - Any license in the list of products with an *issued* date prior to the *replacement date* will be replaced.
  - Any license in the list of products which does not have an issued date, but which has a *start* date prior to the *replacement date* will be replaced.
  - Finally, any license in the list of products with neither an *issued* nor a *start* date will be replaced.
  - EXAMPLE: To replace products “a” and “b”, use: replace=”a b” in the license.

- **Effective Start Date.**
  If *start=dd-mmm-yyyy* is specified in the license, the license cannot be used before the specified date.

- **Timezone Restrictions**

  timezone=timezone-spec

  RLM allows you to specify one or more timezones in which the applications must be running. If a *timezones=timezone-spec* specification is contained in the license, the computer on which the application is running must be set to one of the specified timezones.

  To specify one or more timezones, create a bitmask of the desired timezones, expressed as hours west of GMT:

  Bit 0 - GMT

Bit 1 - 1 hour west of GMT
Bit 2 - 2 hours west of GMT
...
Bit 23 - 23 hours west of GMT (or 1 hour east of GMT)

This bitmask should be represented as a hex number. So, for example, to allow your application to run in the GMT timezone only:

timezone=1

To allow your application to run in timezone 8 (PST):

timezone=100

To allow your application to run in timezones 5-8 (continental USA):

timezone=1E0

The following fields are not used by RLM, but are present to identify licenses, ***or can be used in your application to present to the user:***

- *contract=contract-info*
  contract= is meant to hold the customer's purchase order or software agreement number. This can be used to display to the user to validate a support contract, etc. It is not used by RLM. Maximum of 64 characters.

- *customer=who*
  customer is to identify the customer of the software. This can be an added incentive to keep honest users honest, as it is unlikely that Mega South-East Airlines would want to use a license that was issued to Main St. Bank., for example. *customer* is not used by RLM. Maximum of 64 characters.

- *issuer=who*
  issuer= is used to identify the organization which issued the license. It is not used by RLM. Maximum of 64 characters.

- *_line_item="descriptive_text"*
  The *_line_item* field is used to map a particular product to the item purchased. This field will be logged into the report log at the start when all products supported are logged, so that a report writer can generate reports based on purchased products, as opposed to product names used for licensing purposes. If the descriptive text contains spaces, it should be enclosed in double-quote (") characters. The contents of the *_line_item* field can be modified (or the field can be added) without invalidating the license signature. Maximum of 64 characters.

- *type=type-spec*
  type= is used to identify the type of license. *type-spec* is a string containing one or more of the values:
  - "beta"
  - "demo"
  - "eval"

  (For example, *type="beta eval"* or *type="eval"*. The contents of the license *type* field are retrieved by the rlm_license_type() call (see rlm_license_XXXX()). *type* is not used by RLM.)

The maximum length and types of license fields are as follows:

Field	Type	max data length (excluding keyword=) or value range
isv	string	10 characters
product	string	40 characters
version	string, in the form nnn.mmm	10 characters
exp-date	string of the form dd-mmm-yyyy	11 characters
count	positive integer	2**31 - 1
hostid (single)	string	75 characters
hostid (list)	space-separated, quoted string	200 characters, max of 25 hostids
hostname	string	64 characters
issued	string of the form dd-mmm-yyyy	11 characters
_line_item	string – license administrator defined	64 characters
options	string	64 characters
platform	string	80 characters
start	string of the form dd-mmm-yyyy	11 characters
timezone	int	bitmap with bits 0-23 set
contract	string – unused by RLM	64 characters
customer	string – unused by RLM	64 characters
issuer	string – unused by RLM	64 characters
type	string - consisting of "demo" "eval" and/or "beta"	14 characters

Example

LICENSE reprise write 1.0 permanent uncounted hostid=IP=172.16.7.3 sig=xxxxx

In the example, the *write* product is licensed to a host with an IP address of 172.16.7.3. This is a non-expiring, node-locked, uncounted license.

Note: The keyword "FEATURE" can be used in place of "LICENSE".

RLMTeams line

RLMTEAMS *isvname url* [*username*]

The RLMTeams line, use for the client side of RLM only, is used to identify the correct RLM Teams server for a particular ISV.

isvname is your ISV name
url is the URL of your *RLMTeams* server
username is the (optional) name of the RLMTeams user.

If ***username*** is not specified in the RLMTEAMS line, it must be specified with the RLMTEAMS_USER envirionment variable.

The RLMTeams line is new in RLM v14.2, and HTTPS support must be built into your application for *RLMTeams* to work.

Example:

```
RLMTEAMS demo https://demor.rlmteams.com username@hostname.com
```

Table of Contents

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.

Format of the license file

The license file consists of lines of readable text which describes the actual license grants to your customers. For a complete description of the license file format, see The License File chapter on page 39.

License creation tools

There are 4 ways to create licenses using RLM Embedded:

- rlmgen – GUI license generator
- rlmsign – command-line tool to sign a template license file
- API call – *rlm_sign_license()*
- RLM Activation Pro (an optional product)

rlmgen – GUI license generator

The rlmgen license generator can be used to create licenses interactively. rlmgen is a binary which contains an embedded web server – the UI is presented in your browser. For details on using rlmgen, see the next chapter - Creating Licenses – rlmgen on page 52.

rlmsign

RLM 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.

If you have a back-office sales tracking system, *rlmsign* is the easiest way to integrate license fulfillment. Create an unsigned license file for the sales order, then run *rlmsign* with this license file as it's first parameter. *rlmsign* will sign the license file and make it ready to ship to your customer.

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.

Using rlmsign on Unix

```
% rlmsign license_file [bits-per-character] [-bits bits-per-character] [-maxlen len]
```

Using rlmsign on Windows

```
c> rlmsign license_file [bits-per-character] [-bits bits-per-character] [-maxlen len]
```

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.

You can specify *bits-per-character* positionally as the 2nd argument, or you can use the *-bits bits-per-character* argument any place after the license_file parameter.

The optional *-maxlen len* parameter tells *rlmsign* to set the maximum length of LICENSE lines to the length specified. If this parameter is not specified, the default value of 70 is used. Any field on a license line which would cause the line to go over the maximum length will be placed on a continuation line. If a field can be split across lines (e.g. for fields that are quoted strings), then the field will be split when the maximum length is reached. The maximum length must be between 20 and RLM_MAX_LINE (1024) characters.

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 Embedded also supplies the *rlm_sign_license()* API call to sign a license line in-memory. For details on the *rlm_sign_license()* API call, see Appendix A.

RLM Activation Pro

RLM Activation Pro allows the ISV to give a customer an *activation key* which then allows the customer to retrieve their license from the ISV 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. RLM Activation Pro is an optional product, and details of RLM Activation Pro are in the RLM Activation Pro manual.

RLM Activation Pro is intended to support most common activation scenarios, but in the event that your needs are more complex, or you need help integrating RLM activation with CRM systems, Reprise Software recommends a relationship with one of our 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. Currently, there is one Reprise Product Names in use for RLM Embedded: *rlm_demo* - This product name is used by RLM to enable D*etached Demo*™ licenses for your products.

Table of Contents

Creating Licenses – rlmgen

rlmgen is a GUI tool for use by ISVs to create one-off licenses based on stored product definitions.

rlmgen is a web-based application which allows you to create and edit product definitions for fulfillment, then create individual licenses for in-house or customer use.

rlmgen uses RLM licensing to control access to the two main functions:

- editing the product definition database (rlm_act_admin license)
- creating product licenses (rlm_gen_license license)

The use of these licenses will be described later in this section.

Running rlmgen

Usage: **rlmgen** [optional port#]

Run **rlmgen** from the command line. **rlmgen** uses a web interface which runs on port 6600 by default. If you specify the optional port#, **rlmgen** will use this port number in place of 6600.

Once **rlmgen** is running, point your web browser to "hostname:6600" (http://hostname:6600/home.asp) where hostname is the name of the machine where you run **rlmgen** (or "hostname:alternate port#" if you specified an alternate port#).

rlmgen creates data files for product definitions in the same directory as the binary. It also expects these data files to be in the same directory when it attempts to read them.

rlmgen Licensing and Access Control

rlmgen uses 2 licenses to control access to back-office license operations:

- *rlm_act_admin* license - this license enables the creation and editing of product definitions.
- *rlm_gen_license* license - this license enables the creation of individual licenses using the "Create License" button.

When you install the RLM kit, nodelocked-uncounted licenses (locked to hostid ANY) are created for both of these licenses so that you can get started. If you do not care about access control to **rlmgen**, you need do nothing further. On the other hand, if you do care about access control to these functions, you should edit the licenses created during the install procedure to limit access to **rlmgen**. In any case, you should make sure that the license file is present in the same binary as the **rlmgen** binary if you move **rlmgen** to a different directory.

Since **rlmgen** is a web-based application, you will need to issue licenses for *rlm_act_admin* and *rlm_gen_license* for the IP addresses of the machines you would like to be able to perform these activities. Edit these licenses for your particular network.

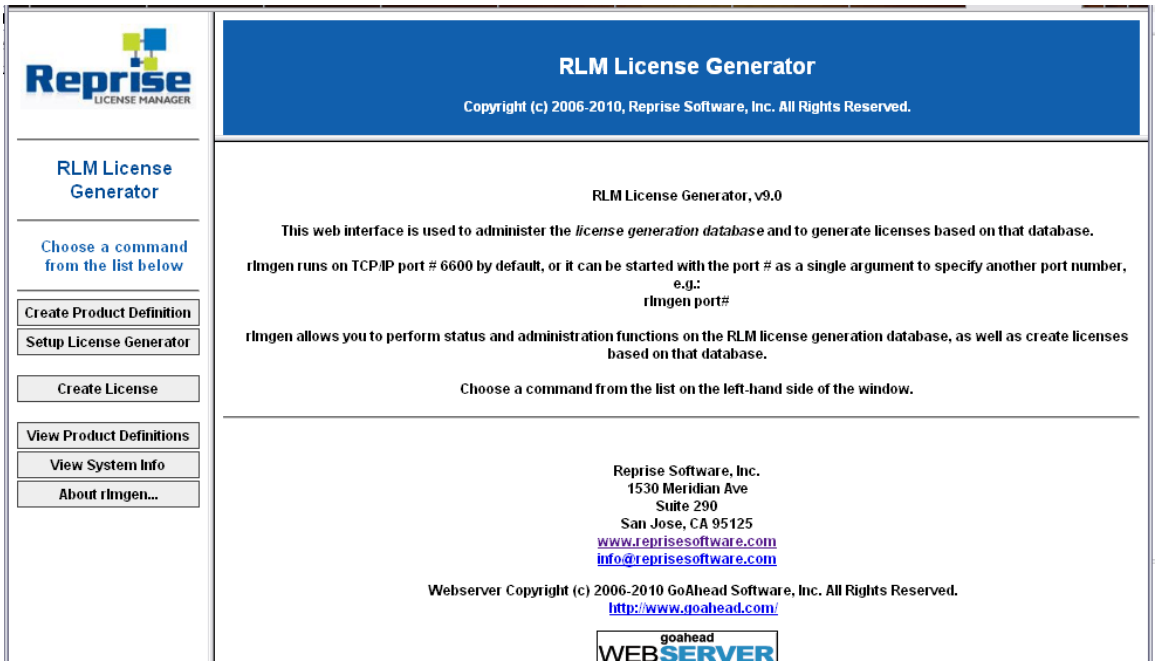
Using rlmgen

There are 2 main activities you will perform with **rlmgen**:

- initial setup of your product definitions
- generating individual licenses

When you run **rlmgen**, the application will start a web server on port 6600 (or an alternate port if you specify on the **rlmgen** command line). Next, point your browser at port 6600 on the machine where you ran **rlmgen**.

Your browser will display a page that has a title area at the top, along with a column of command buttons along the left, and a main display area on the bottom right hand side. The startup screen is shown below:



The command buttons are arranged in 3 groups, top to bottom:

- setup commands - "Create Product Definition" and "Setup License Generator"
- license generation commands - "Create License", and
- Database viewing/updating commands - "View Product Definitions", "View System Info", and "About rlmgen..."

Each of these commands will be described in the sections which follow.

Setup Commands

The 2 setup commands are:

- **Create Product Definition**

- **Setup License Generation**

These commands are used to set up the database for license generation. Everything in rlmgen is driven from *product definitions*, so you should create at least one product definition first. Product definition allows you to specify a name for the "product" which is then associated with a license product name, version, and several other license parameters.

The **Setup License Generation** form allows you to specify which optional RLM license parameters will be presented in the GUI for specification. Any parameters checked on this form will appear in the license creation screens. If you specify some of these parameters as part of the product definition (in the "other license parameters" field), they will not appear in the list of checkboxes in **Setup License Generation** (since they are already specified for the product). You should leave any parameter which you do not want to put into a generated license unchecked in this form.

License Generation Commands

There is a single license generation command **Generate License**, which brings up a form that allows creation of a single license from parameters specified in the forms.
The data appearing in the license creation forms is customized by both the product definition as well as the **Setup License Generation** form.

Database Viewing/Updating Commands

These commands are:

- **View Product Definitions**
- **View System Info**, and
- **About rlmgen..**

These commands all allow viewing of the product definition database. The first command also will allow you to edit product definitions, by pressing the "Edit" or "Disable" button at the end of the row in the display. Product definitions which are disabled are not deleted, but will appear in gray in the list, and can be re-enabled later.

View System Info displays information about the system where **rlmgen** is running, including the RLM platform, RLM version, and hostids for this system.

Finally **About rlmgen...** displays information about the rlmgen program itself.

Table of Contents

End User Installation

When your customer receives your product, they need to do a few things in order to set up the licensing. If they are familiar with popular license managers, these installation steps should be quite familiar.

The steps the license administrator needs to perform to install licensing, in addition to installing your application, are:

- Install license file
- Optionally set up environment for users to find the license file when running your application.

In addition, your users would also want to know:

- Where to find the license administration tools
- You might also want to provide them with the RLM License Administration manual (or a link to the manual on the Reprise Website). If you provide a link to the Reprise website, your customers will always have the manual for the latest released version of RLM (which always includes information about all RLM versions, including which version certain features appeared).

Reprise Software's Recommended Software Installation steps

Installing your product with RLM Embedded should be very straightforward, and should require no configuration of environment variables, etc.

On the machines where your application is going to run, place the license file in your product hierarchy. For nodelocked licenses, this should be the actual, signed license file, and nothing needs to be done to this license file.

Nothing in this set of recommendations requires the use of environment variables, and the install-time editing of license files is kept to a minimum (No editing of license files for nodelocked licenses, and only the server hostname needs to be set on the client side for floating licenses).

Details:

During development:

- establish a directory in your installed product tree for license file(s). This could be the same directory where your product is installed.
- Pass the directory from the step above as the first argument to rlm_init().

When you first ship the license:

- Put the actual licenses into the license file. Install in the default directory. You're done.

If you ship new, additional licenses to your customer:

- Put the new license file in the same directory as the old one. You're done. RLM Embedded will read all the license files in this directory.

Table of Contents

Pre-Release Checklist

With RLM Embedded, 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 Embedded 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 Embedded Integration section and ensure that your product and installation are well-behaved.
- If you use the RLMID1 option, add documentation on installing and using the device:
  - Ensure that **INCLUDE_RLMID1** is defined in your *rlm_isv_config.c* file if you plan to create node-locked licenses locked to an rlmID1 device.
  - Windows update will perform the required steps for internet-connected systems. However, for your users who have licensed systems not connected to the internet, your user will need to run the driver setup utility. Include the driver installer located at: http://www.reprisesoftware.com/drivers/rlmid1.zip
  - On Linux, the driver installation download is at "http://sentinelcustomer.safenet-inc.com/sentineldownloads/". Select one of the "Sentinal HASP ... Runtime Installer" options, where the operating system in the 3rd column is Linux. The installation is available in RPM, and compressed tar formats. The installer starts the driver, and sets up rc scripts so that the driver is started when the system boots.
  - Include instructions for your license administrators to install the hardware key.
- A good practice is to include a folder for licenses in your installed product folder tree. Then any license you ever issue - an expiring demo license or a production nodelocked license - goes in one or more .lic files in that licenses folder. Given that you have passed the path to that directory to rlm_init(), your application will always be able to find the licenses.

Table of Contents

Section 3 – Advanced Topics

This section of the manual contains topics that may be of use if you are doing a more advanced implementation of licensing.

Upgrading to a New Version of RLM Embedded

If you have previously integrated RLM Embedded into your product and wish to upgrade to a new RLM Embedded version, follow these steps:

- First, Download the new RLM kit from the Reprise website - see details in the Installing RLM Embedded chapter, above.
- Then, unpack the kit and install. See details in the Installing RLM Embedded chapter, above.
- Next, copy the following 3 files from your old kit:

rc/rlm_pubkey.c - copy this file - do not use a new public/private key set from the installation
src/rlm_privkey.c - copy this file - do not use a new public/private key set from the installation
src/rlm_isv_config.c - copy this file - unless you want to change the configuration of RLM Embedded for this version

- Edit the following 2 files in the new kit:

src/license_to_run.h - modify this to install the new RLM Embedded license you received from Reprise Software
platform/makefile - modify the ISV= line to contain your ISV name (always required on Windows - this step is done as part of the INSTALL process on Unix)

- Finally, run **make** (or **nmake**) in the kit binary directory, and your RLM Embedded libraries rlmsign binary and activation binaries are ready to use.

Table of Contents

Using RLM Embedded with Languages other than C/C++

If the language you are implementing your application in can link to the RLM static library then that is the recommended approach. Other languages must use the RLM dynamic library, rlm<vvmm>.dll on Windows or rlm<vvmm>.so on Unix/Linux. <vvmm> indicates RLM version, eg, the DLL for v11.0BL2 is rlm1102.dll.

In all cases, you will need to download the RLM SDK and build it with the correct C compiler. On Windows, you can download Microsoft Visual Studio Express for free and use it to do the RLM build.

The following sections explain how to use RLM Embedded with various languages.

Using RLM Embedded with Fortran

The examples/unsupported directory on the RLM SDK contains a Fortran interface for RLM.

Using RLM Embedded with MinGW

You need to use the RLM dll with MinGW, as the RLM library is compiled with Visual C++ and those object modules can't coexist with MinGW object modules in the same executable. Link your application with rlm<vvmm>.lib, and at runtime make sure that rlm<vvmm>.dll is in your PATH. <vvmm> indicates RLM version, eg, the DLL for v11.0BL2 is rlm1102.dll.

Here is a makefile example, which builds the demo client rlmclient.exe with MinGW. Here rlmclient is analogous to your application. (Note that using gcc to perform the link instead of ld means that gcc finds all the right system libraries rather than you having to enumerate them on the ld command line):

```
rlmclient.o:        ..\examples\rlmclient.c
        gcc -I..\src -o rlmclient.o -c ..\examples\rlmclient.c

rlmclient.exe:      rlmclient.o rlm1102.lib
        gcc -o rlmclient.exe rlmclient.o rlm1102.lib
```

Using RLM Embedded with Visual Basic (outside .NET)

For information on integrating RLM with Visual Basic 6 applications, see http://www.reprisesoftware.com/tutorials/Using_RLM_with_Visual_Basic.pdf . The following material covers later versions of Visual Basic.

Visual Basic provides a means to make calls to functions in a DLL. This can be used to call RLM functions in rlmVVVV.dll. This is done by declaring the RLM functions you need to use in Visual Basic's "Declare Function" statements, identifying the location of the DLL and how to pass each argument. There is a good technical article on how to do this at http://support.microsoft.com/kb/106553/EN-US/ - see sections 2.0 and 2.1. (Note: VVVV signifies the RLM version, such as 943 for 9.4BL3 or 1002 for 10.0BL2)

Write Declare Function statements for the RLM functions you want to call. To figure out the mapping between basic data types and C data types, first look at src\license.h on the SDK, to see how each function is declared in C, the use the corresponding datatype and calling convention in Basic. Some guidelines:

- Where an RLM function returns some sort of handle, like RLM_HANDLE or RLM_LICENSE, declare the function in Basic "As IntPtr"
- Where an RLM function returns an int, declare the function in Basic "As Integer"
- Where an RLM function argument is a handle type (RLM_HANDLE, RLM_LICENSE, etc), pass it as "ByVal ... As IntPtr"
- Where an RLM function argument is type "char *", pass it as "ByVal ... As String"
- Where an RLM function argument is type "int", pass it as "ByVal ... As Integer"

Here is a simple example Visual Basic program that checks out a license and checks it back in:

```
Module Module1
    Declare Function rlm_init Lib "rlm.dll" (ByVal path As String, ByVal appPath As String, ByVal
license As String) As IntPtr
    Declare Function rlm_stat Lib "rlm.dll" (ByVal handle As IntPtr) As Integer
    Declare Function rlm_checkout Lib "rlm.dll" (ByVal handle As IntPtr, ByVal name As String, ByVal
version As String, ByVal count As Integer) As IntPtr
    Declare Function rlm_license_stat Lib "rlm.dll" (ByVal license As IntPtr) As Integer
    Declare Function rlm_checkin Lib "rlm.dll" (ByVal license As IntPtr) As Integer
    Declare Function rlm_close Lib "rlm.dll" (ByVal handle As IntPtr) As Integer

    Sub Main()

        Dim response As String
        Dim path$ = "."
        Dim nullstring$ = ""
        Dim handle As IntPtr
        Dim license As IntPtr
        Dim product$ = "test1"
        Dim ver$ = "1.0"
        Dim stat As Integer

        handle = rlm_init(path$, nullstring, nullstring)
        stat = rlm_stat(handle)
        If stat = 0 Then
            license = rlm_checkout(handle, product, ver, 1)
            stat = rlm_license_stat(license)
            If stat = 0 Then
                Console.WriteLine("Checkout succeeded, hit CR to check in...")
                response = Console.ReadLine
                stat = rlm_checkin(license)
            Else
                Console.WriteLine("rlm_checkout error " + stat.ToString("d"))
            End If
            stat = rlm_close(handle)
        Else
            Console.WriteLine("rlm_init error " + stat.ToString("d"))
        End If

        Console.WriteLine("Hit CR to exit...")
        response = Console.ReadLine
    End Sub

End Module
```

Using RLM Embedded with .NET

Overview

RLM Embedded provides a solution for .NET developers who want to use RLM Embedded to license their applications. It consists of a simple Interop layer that defines the RLM Embedded functions in .NET terms, and a DLL containing the native code. Here is the high-level overview of how to use this capability:

- Install, configure and build a Windows RLM SDK. This provides the utilities, and the actual RLM Embedded code packaged in a DLL.
- Build the VS project "Reprise" in the *dotnet* folder on the SDK.
- Add calls to the RLM Embedded methods to the .NET application to be licensed.

Building the RLM Embedded .NET package

If you have VS2005 or later, simply double click on *dotnet\Reprise\Reprise.sln* to open the project in Visual Studio, then build the project. You can build Debug or Release or both. If you have a prior version of Visual Studio, then create a project for the RLM Embedded .NET code manually, copy *dotnet\Reprise\Reprise\RLMInterop.cs* into it, and build.

Running the Example Program

The example program expects the example license file from the SDK (example.lic) to be correctly signed and to be available. The example program will check out v1.0 of the license "test1". Here are the steps to run the example program::

- Open *dotnet\RLMTest\RLMTest.sln* in Visual Studio 2005 or later. If you have an earlier version of Visual Studio, create a new project for RLMTest, as described above for Reprise.
- Build the project, either Release or Debug or both, but in the same configuration(s) as you built the Reprise project.
- Copy rlmVVVV.dll, which contains the actual RLM code, from your platform folder (x86_w* or x64_w*) to some folder which is on your PATH, so that it can be found at runtime by the application. (Note: VVVV signifies the RLM version, such as 943 for 9.4BL3 or 1002 for 10.0BL2)
- Copy example.lic, which is the signed license file, from your platform folder (x86_w* or x64_w*) to the same folder containing the application.
- Run the application. It opens a command window for it's output, which will look like this if it runs successfully:

```
rlm_init successful
hostid of this machine is <your machine's hostid>

test1
version 1.0
expiration permanent
test2
version 1.0
expiration permanent
test3
version 1.0
expiration permanent
rlm_license_center
version 7.0
expiration permanent
rlm_act_admin
version 7.0
expiration permanent
```

```
        rlm_act_view
        version 7.0
        expiration permanent
        rlm_gen_license
        version 7.0
        expiration permanent
        rlm_roam
        version 1.0
        expiration permanent
        checkout of test1 OK
        attributes of test1
        expiration: permanent
        days until expiration: 0
        checkout of not_there failed: License server does not support this product (-18)
```

If the checkout of test1 fails, it is likely that either the license server is not running, or rlmVVVV.dll cannot be found.

Integrating RLM Embedded .NET into your Application

This manual serves as a description of the routines available for a license application to call and how they behave. Refer to *dotnet\Reprise\Reprise\RLMInterop.cs* for the argument types and return value types in the .NET world.

Include a:

```
using Reprise;
```

statement with any classes that invoke RLM, and precede RLM function names and constants with "RLM.", for example, "RLM.rlm_checkout". See the example program *RLMTest.cs* for an example.

You will need to include a reference to RLM in your application's project. The object to reference is: *<platform>\Reprise\Reprise\bin\<Debug or Release>\Reprise.dll*

Several RLM Embedded functions are not supported in RLM Embedded .NET. They are:

- rlm_isv_cfg*
- rlm_sign_license
- rlm_add_isv_hostid
- rlm_add_isv_compare
- rlm_add_isv_multiple
- rlm_all_hostids
- rlm_auto_hb
- rlm_act_refresh

Table of Contents

Debugging Licensing Problems in the Field

In order to diagnose a licensing problem in the field, you will need some information from your customer. RLM Embedded v8.0 and later provides client-side diagnostics which show the application environment and local licenses available for checkout. In addition, RLM Embedded v9.0 adds product debugging information.

Client-side Diagnostics

Built into every RLM Embedded v8.0 (and later) client is the ability to output environmental information about the application's use of RLM Embedded. To enable this, your customer simply sets the environment variable **RLM_DIAGNOSTICS** to the name of a file, then runs your application. Once you call rlm_init(), RLM will write diagnostic information to the file name specified. (Note that if you simply set **RLM_DIAGNOSTICS** without a value, the output will be sent to standard out - which may not be what you want).

The resulting output will give the following information:

- time the program was run
- working directory
- relevant environment variables
- list of RLM's idea of the hostids on the machine where the application was run (including your ISV-defined hostids)
- the license files in use, in the order RLM will use them (can be re-ordered from your normal list if RLM_PATH_RANDOMIZE is set).
- the parameters you used in your call to *rlm_init()*
- a list of all local licenses which can be checked out.

An example of this information is contained here:

```
RLM Diagnostics at 10/09/2009 15:40

Environment:

    Hostname: paradise
    Working directory: /user/matt/rlm/test
    RLM platform: x64_s1
    OS version: 5.10

        HTTP_PROXY=<not set>
        RLM_CONNECT_TIMEOUT=<not set>
        RLM_EXTENDED_ERROR_MESSAGES=<not set>
        RLM_LICENSE=2700@paradise:a.lic:b.lic
        RLM_NO_UNLIMIT=<not set>
        RLM_PATH_RANDOMIZE=<not set>
        RLM_PROJECT=<not set>
        RLM_QUEUE=<not set>
        RLM_ROAM=1
        RLMSTAT=<not set>
        REPRISE_LICENSE=<not set>

    RLM hostid list:

        1d8bbd06 ip=172.16.7.13

License files:
        2700@paradise
```

```
        a.lic
        b.lic

rlm_init() parameters:
        1: <empty>
        2: client3
        3: <empty>

Local licenses which can be checked out


    In license file a.lic
          (no server)

        test1 v1.0 OK
        test2 v1.0 error:-5
        test2 v1.0 error:-3
```

In this example, you can see that a checkout of "test2" would not succeed with a nodelocked license in license file *a.lic*, because the first test2 license has a bad license signature (error -5, RLM_EL_BADKEY) and the second test2 license has expired (error -3, RLM_EL_EXPIRED).

You can also see that this application will attempt a checkout from the license server running on node **paradise** at port **2700**, if none of the local licenses will satisfy the checkout request.

Product Debugging Information

Beginning in RLM Embedded v9.0, your application will be enabled to output diagnostic information about any or all product names. In order to do this, set the environment variable ***RLM_DEBUG*** as follows:

- RLM_DEBUG set to an empty value – show information about all products
- RLM_DEBUG set to a string – show information about the product specified

This information can be obtained from the new rlmdebug utility (part of rlmutil), or directly from your application. If the RLM_DEBUG environment varilable is set, the debugging information will be output to stdout at the end of the *rlm_init()* call. For use of rlmdebug (which does not require the RLM_DEBUG environment variable), see the RLM License Administration Manual.

Note that the most accurate information will be obtained from your application, since the exact license file path used by the application will be available to the rlm debugging routines. The stand-alone utility cannot know about default license files and paths which you set in your *rlm_init()* call. Please note that RLM_DEBUG will only report on licenses which are present in local license files. In other words, if you have a license path like “5053@server”, RLM_DEBUG will report on whether the server is up, but it will not report on individual licenses served by that server.

For example, with RLM_DEBUG set to an empty string:

```
% setenv RLM_DEBUG
```

The following (sample) output is displayed:

```
  RLM DEBUG for all products
In license file: ../rlm/z.lic (5555@paradise):
Product: test1, ISV: reprise, Floating
```

```
   Product: test2, ISV: reprise, Floating
   Product: test3, ISV: reprise, Floating
   Product: rlm_roam, ISV: reprise, Uncounted
   Product: testr1, ISV: reprise, Floating
   Product: testr2, ISV: reprise, Floating
   Checking server machine "paradise" ... server UP
   Checking RLM server at port 5555 ... server UP

   In license file: a.lic:
   Product: test, ISV: reprise, Single

   8 product instances found
```

On the other hand, with RLM_DEBUG set to the name test:

```
% setenv RLM_DEBUG test
```

The following (sample) output is displayed:

```
RLM DEBUG for product "test"

   In license file: ../rlm/z.lic (5555@paradise):
   Checking server machine "paradise" ... server UP
   Checking RLM server at port 5555 ... server UP
   No matching products found in license file

   In license file: a.lic:
   Product: test, ISV: reprise, Single

          1 product instances found
```

Table of Contents

Alias Licenses

Alias licenses are licenses that define themselves in terms of another license.

For example, an application could request a license for product **write**. If this were a *normal* license, the product **write** would appear in the license file (if the request succeeds). On the other hand, if this were an *alias* license, the product **write** would appear in the license file without a count, but with a specification of one other product which is used to satisfy the request. When the client encounters a request for an *alias* license, it uses the other product specified in the license to satisfy the request, rather than the originally-requested product. This other license is called the *primary* license. The primary license must be another nodelocked license.

A alias license differs from a normal license in a few significant ways:

- The count field contains one of the keyword **alias** rather than an integer, **uncounted**, or **single**.
- The license has an alias spec: alias=**"<prod ver count>"**
- The only optional parameters on an alias license which are used by RLM are the start date, and the hostid. All other optional parameters are ignored.

Example of an *alias* Licenses

When a product is specified as an *alias* license, requests for that product are turned into requests for the *primary* license specified in the **alias=** part of the license. For example, consider this license for product **test** (*primary* license **dollars**):

```
LICENSE reprise test 1.0 permanent alias sig=xxxx alias="<dollars 2.0 5>"
LICENSE reprise dollars 2.0 permanent hostid=abcdef01 sig=xxxx
```

A request for the product “test”, v1.0 will check out “dollars”, v2.0

The License Count Keyword

In an *alias* license, the count keyword is “alias”.

If you want the alias license usable only on a single host, include the “hostid=xxx” keyword in the alias license itself.

The *alias*= keyword

In an *alias* license, the *alias*= keyword specifies the *primary* license which is checked out in response to a request for the *alias* license itself. Specify only one license to be checked out. **Although the syntax processing is the same as for token-based licenses, only the first product specified will be used**. These licenses also cannot be *alias* licenses themselves. The format is:

```
alias="<product1 ver1 count1>"
```

The request for the one of the original license turns into a checkout of *count1* of *product1, ver1*

Nesting *alias* licenses

Alias licenses cannot be nested.

Restrictions on *alias* licenses

All *alias* licenses are processed by the client, so there can be no floating *alias* licenses, in fact, the license server completely ignores *alias* licenses.

Use cases for alias licenses

Perhaps the most compelling use of an alias license is in conjunction with Activation Pro. If you sell several different product bundles, your options for doing that are either to issue independent licenses for all the products in the bundle, or to use an options= keyword in a single RLM license and decode the options in your product. The second approach has a couple of disadvantages: (1) you have to process the options yourself, and, more importantly (2) it becomes nearly impossible for your customer to select which bundle they want to check out. The first approach (separate licenses) is easier for everyone, until you get to issuing activation keys for the bundle. No one wants to issue N independent activation keys. Using alias licenses allows you to avoid this. Let's say you have 4 products: a, b, c, and d, which you sell as 3 different bundles: x, y, and z. With alias licenses, you can do the following, assuming you want a and b in bundle x, a, b, and c in bundle y, and a, b, c, and d in bundle z:

1. create static definitions of the bundles, as follows:

   ```
   alias a to x
   alias b to x

   alias a to y
   alias b to y
   alias c to y

   alias a to z
   alias b to z
   alias c to z
   alias d to z
   ```

   These alias definitions can be shipped with your product, or updated when you have new bundle definitions. The point is that they are the same for all your customers, and they are independent of your code.

2. When your customer purchases bundle “y” for example, issue them an activation key for product y. Now RLM will allow them to check out a, b, and c, based on those alias lines, but not d.

A second use of an alias license is to allow alternate licenses to satisfy a request for a product. To use the familiar example, if product **write** checks out a *write* license, the addition of an *alias* license for *write* mapping it to *office* would allow an *office* license to be used in the case where no *write* licenses are

available. Even though the *office* license is a more expensive license, the customer is allowed to continue working by consuming the more expensive *office* license. Several *alias* licenses can be used in this way, and the order of the licenses in the license file will determine the order that alternate checkouts are attempted.

Table of Contents

ISV-defined Hostid Processing

RLM provides the ability to extend the native set of hostids by using your own routines to obtain host identification which is unique to you. There are 2 methods to do this – the older, deprecated "isv-defined hostid", and the newer "ISV=" hostid type.

If you use the deprecated isv-defined hostid:

- Your customers cannot use the generic rlmhostid tool, you must write and ship your own tool.
- You cannot use the standard Activation Pro license generator, you must build a custom generator - which may involve licensing an additional RLM platform.

Beginning in RLM v11.3, Reprise Software recommends using the new ISV= hostid type, which uses the ISV-defined string as set by the rlm_set_environ() call. The advantages and disadvantages of doing this, over the older ISV-defined hostid code are:

Advantages:

- no custom code to write (other than getting the string itself)
- you can use an Activation Pro generator settings file and avoid building a custom generator

Disadvantages (compared to the old isv-defined hostid):

- only one hostid of this type is supported on a system
- the hostid comparison code is always a case-insensitive strcmp() function

To use this hostid type, do the following:

1. Determine the hostid on your system (in your application).
2. Call rlm_set_environ(....., your-hostid), immediately after your call to rlm_init()
3. Use "isv=your-hostid", as the hostid for your license.

If after reading this you still want to write code to create an isv-defined hostid, please contact Reprise Software**.**

Table of Contents

Shipping Your Product as a Library or a Plugin

In some cases, your product might be a library/DLL/Shared Object which is linked into other programs. If these other programs use RLM as well, you will need to do something to avoid name collisions between your copy of RLM and the other program's copy of RLM (the other program may use a different RLM version, for example).

The technique for accomplishing this is a bit different for Windows and Linux systems.

Windows

Create a DLL that contains the code for your product as well as the RLM Embedded code. When you create the list of exports from the DLL, don't include any RLM Embedded functions.

If for any reason you need to keep your DLL and the RLM DLL separate, you should take these additional steps to avoid collisions with other ISVs' versions of the RLM DLL which may be present at runtime:

- Modify the makefile in the platform directory such that the name of the DLL that gets built includes your ISV name. For example, if your ISV name is "xyz", modify these lines in the makefile:

  ```
  DLL = rlm$(VER).dll
  DLLLIB = rlm$(VER).lib
  ```

  to read:

  ```
  DLL = xyz_rlm$(VER).dll
  DLLLIB = xyz_rlm$(VER).lib
  ```

- Run nmake in the platform directory to build the DLL under the new name
- Change any code that references the DLL by name to reflect the new name. An example is RLMInterop.cs - the C# interface to RLM.
- Link your DLL against <new DLL name>.lib
- Update your installer to include the RLM DLL, and install it in the same location as your product's DLL.

This will ensure that at runtime even if multiple RLM DLLs are present on the machine, your code will invoke the correct RLM DLL.

Linux

Create a shared object (.so) that contains the code for your product as well as the RLM Embedded code. When you link your shared object, include the following on the command line:

```
-Wl,--version-script=file
```

(Note: the character after the uppercase W in the command above is a lowercase l, as in "license" If you create the .so file with ld instead of cc, then just use the -version-script=file option.

file should contain:

```
{
        global:
                function1;
                function2;
                ...
                functionN;

        local:
                *;
};
```

function1, ***function2***, etc, are your functions that can be called from outside the shared object.

The advantage of this technique is that all the RLM Embedded symbols will be redefined as local symbols in your .so file.

Alternately, you can specify the ***-Bsymbolic*** switch to ld, as follows:

ld -share -Bsymbolic your-object-files.o rlm.a

The -Bsymbolic option tells the loader to bind references to the global symbols in the rlm library (and any other global symbols in your object list) to the definitions within your shared object rather than using previous definitions from other shared objects. This works, however all the RLM Embedded objects will remain globals in your .so file.

Table of Contents

Internet Activation

Overview of RLM Activation Pro

RLM Activation Pro allows you to deliver an **activation key** to your customer, and when they are ready to use your product, a transaction with the activation server allows the license to be fulfilled without manual intervention. When using activation, there is no need for you to get your user's hostid information - this is transmitted to the activation server automatically.

A typical scenario would be that your customer runs the product on the desired machine, and if the license had not been fulfilled earlier, the product asks for an activation key. Once the activation key is supplied, the license is retrieved transparently. From this point on, the product runs with its license in place.

RLM Activation Pro allows you to deliver *rehostable licenses* as well, by creating a license that is locked to a hostid that can be removed from the target system when a rehost is requested.

RLM Activation Pro is an optional part of the RLM product. For complete details on RLM Activation Pro, see the *RLM Activation Pro Manual*.

Table of Contents

Virtualization

RLM Embedded provides capabilities to enable and/or restrict the usage of your application in virtual environments.

By default, RLM allows applications to run in virtual environments.

You can restrict application usage in virtual environments by using the "disable=vm" keyword in the license. See the **disable** keyword in the LICENSE Linesection on page 40 in The License File for more information.

Table of Contents

Securing Your Application

No software is 100% resistant to a talented and determined hacker, but there are steps that RLM Embedded takes to thwart hackers, and there are steps your application can take as well. The best possible protection is available from 3rd party application hardening providers, who are listed on the Partners page at www.reprisesoftware.com. Short of that, here are some techniques that hackers use to circumvent licensing in applications, and some steps you can take to thwart them.

DLL / shared object spoofing

If your application uses the RLM Embedded DLL (or Unix shared object) instead of linking to the static RLM Embedded library, a hacker may substitute one of his own where *rlm_checkout()* always succeeds. You can detect this in your code by attempting a license checkout that should never succeed. For example, check out a license that your code never uses, or an impossibly high version number or quantity. If it succeeds, then there's a spoofed DLL or shared object present. If you detect a spoof, you can elect not to fail immediately. Set a fail flag and check it later and fail then. This makes it harder for the hacker to correlate the failure with the spoof check.

Public key injection

This is a technique whereby the hacker creates his own public-private key pair, and patches his public key in place of the ISV's public key in the application. Then the hacker can sign any license using his own key pair and have the application authenticate it correctly. There are two techniques that can be used to thwart this:

- Add extra calls to rlm_init() specifying NULL in the first two arguments, and passing a license in the third argument that you have signed (using your authentic key pair). Then attempt to check it out. If a different public key has been injected into the application, the checkout will fail with a bad signature (-5) error. To make it more difficult for a hacker to patch in his own license to your extra rlm_init() invocations, don't store the license in a single string in your code. Break it up into small bits and assemble it at runtime for passing to rlm_init(), or store it in an encrypted form and decrypt it on the fly.

- Verify that the public key is in fact yours. You can do this by calling *_rlm_get_pub()* in your application, which returns the bytes that make up the public key. The signature for the function and the bytes of the key are in src/rlm_pubkey.c. Then check some number of bytes in the key returned against what they should be.

You can use either one of these techniques or both. As with DLL spoof detection, consider delaying failure if you detect an injected public key.

Decompilation of .NET, recompiling without licensing

Because the compiled format of .NET is much closer to a high-level language than traditional object code, decompilers exist for those formats that produce very readable source code. With decompiled source, the hacker can see where the licensing methods are called, remove them, and recompile what amounts to a version of the application without licensing. Tools called "obfuscators" can be used by ISVs to rearrange the application's logic and obscure method and variable names such that the decompiled code is very difficult to understand. Reprise does not recommend any particular obfuscators, but there are several available.

How RLM Clients Find the License

There are a number of ways for RLM clients to locate a license. These include:

- a license filename passed as argument 1 to rlm_init()
- a license filename contained in a directory passed as argument 1 to rlm_init()
- a license filename in the application binary directory, as passed in argument 2 to rlm_init()
- a license file referenced from the RLM_LICENSE or ISV_LICENSE environment variable.
- a port@host specification contained in the RLM_LICENSE or ISV LICENSE environment variable

  Note that Reprise Software recommends placing your license file in the directory with your binary, and passing this directory name (usually ".") as the 2nd argument to *rlm_init()*.

Table of Contents

Wide Character Support

Because of the way Unix and Linux systems store directory and filename information, that is, in UTF-8, there has never been an issue with running RLM Embedded in wide-character environments on those systems. On Windows however, where the operating system stores path information in wide-characters, RLM Embedded has to be wide-character-aware.

Starting in v11.0, if an application passes paths to rlm_init() containing wide characters (wchar_t or WCHAR strings), it must first convert those paths to UTF-8 before passing them to RLM Embedded. RLM Embedded stores them internally as UTF-8, and converts back to wide characters before using them in filesystem operations. In this way, RLM Embedded clients can be installed on wide character paths and work correctly.

On Windows platforms, if the paths your application would pass to rlm_init() in the first and second parameters are Unicode wide characters (wchar_t or WCHAR), you must first convert them to UTF-8. The Win32 function WideCharToMultiByte() can be used for this conversion.

Table of Contents

Using Activation Pro with HTTPS

Beginning in RLM v14.1, Activation Pro supports the HTTPS prototol as well as HTTP.

You will need to ensure that you can handle the new RLM_EH_WEBS_NOSUPP and or RLM_EH_NOHTTPSSUPPORT status returns, which will be returned on certain RLM API functions that are not supported with web services. See the last section in this chapter for more information.

If your company does not use Activation Pro, you can ignore this chapter.

How to use HTTPS with Activation Pro

First of all, HTTPS transport is only supported in the linux, Mac and Windows versions of RLM. Any other platform will return RLM_EH_NOHTTPSSUPPORT if you specify an HTTPS URL to Activation Pro (or your URL redirects to an HTTPS URL).

To use HTTPS with Activation Pro, follow these instructions:

- **For Linux and Mac:**
  - link your RLM application with the web services http module *rlm_msgs_http.o*. Place this object file before the rlm library in your link line. Consult the build rules for *rlmclient_http* in the makefile for an example.
  - You must ensure that every system where your application runs has the CURL libraries installed. (These libraries are always installed in our experience on Mac). If the libraries are not installed, you will get a message similar to this when you run your application on linux:

```
your-program-name: error while loading shared libraries: libcurl.so.4: cannot
open shared object file: No such file or directory
```

On Ubuntu linux, this is solved with the following command:

```
% sudo apt-get install libcurl-openssl-dev
```

- **For Windows:**
  - link your RLM application with the web services http module *rlm_msgs_http.obj*. There are 4 different variants of *rlm_msgs_http.obj* for Windows, corresponding to the 4 different object formats (/MD, /MT, /MDd, and /MTd). They are *rlm_msgs_http_md.obj*, etc. When building your application, select the appropriate one for the style of build you're doing and place it on your LINK command line *before* the RLM library. See the build rule for *rlmclient_http.exe* in the makefile for an example.
  - Unlike on Linux, using the HTTPS capability in RLM does not require any external library dependencies. All the low-level CURL/HTTPS support is in the RLM client library. Of course it is linked into your application only if you use *rlm_msgs_http_*.obj*.

- RLM HTTPS support on Windows does not work on Windows/XP and Windows Server 2003. We believe it will work on Windows/Vista and Windows Server 2008 but we have not tested it. We have tested on Windows 7 and later. If you run an HTTPS-enabled application on XP/Server 2003 machine, you will get the RLM_EH_WS_NOSUPP error.

Once your application is built with the correct support, and the CURL libraries are installed on the target system, using HTTPS is simply a matter of specifying an https:// URL rather than http://

That's all there is to it. Everything else remains the same; the RLM client library switches to use HTTPS transport when appropriate.

Table of Contents

Section 4 – Reference Material

Appendix A – RLM Embedded API

This appendix lists all the RLM Embedded API calls in alphabetical order.

To call any of the functions in the RLM Embedded API, you need to include the RLM Embedded header file “license.h”:

```
#include "license.h"
```

RLM_HANDLE

RLM_HANDLE is the main handle in RLM Embedded. Your program needs to call *rlm_init()* to get a handle; this only needs to be done once. This handle (or an RLM_LICENSE handle) is passed to all the RLM Embedded api calls.

RLM_LICENSE

RLM_LICENSE is the license handle in RLM Embedded. This handle is returned from the *rlm_checkout()* call, and is passed to the *rlm_license_stat()*, *rlm_get_attr_health()* and *rlm_checkin()* calls.

N.B. RLM_LICENSE is also the name of the environment variable for specifying the license file path.

Core API – these 8 functions provide all basic licensing operations needed for most applications:

```
rlm_init() - initalize licensing operations with RLM Embedded.
rlm_close() - Terminate licensing operations with RLM Embedded.
rlm_checkout() - Request a license.
rlm_checkin() - Release a license.
rlm_errstring() - Format RLM Embedded status into a string.
rlm_stat()- Retrieve RLM_HANDLE status.
rlm_license_stat() - Retrieve RLM_LICENSE status.
rlm_get_attr_health() - Check license status.
```

Advanced API – most applications will need few, if any, of these calls:

```
rlm_activate()
rlm_act_info()
rlm_act_destroy_handle()
rlm_act_fulfill_info()
rlm_act_info()
rlm_act_keyinfo()
rlm_act_keyvalid()
rlm_act_new_handle()
rlm_act_rehost_revoke()
rlm_act_set_handle
```

rlm_add_isv_hostid()
rlm_add_isv_hostid_compare()
rlm_add_isv_hostid_multiple()
rlm_detached_demo()
rlm_detached_demox()
rlm_errstring_num()
rlm_get_attr_lfpath()
rlm_get_rehost()
rlm_hostid()
rlm_all_hostids()
rlm_all_hostids_free()
rlm_license_XXXX()
rlm_products()
rlm_putenv()
rlm_set_attr_req_opt()
rlm_set_attr_reference_hostid()
rlm_sign_license()

Namespace

- All RLM client library functions have names beginning with rlm_ or _rlm_.
- All defined constants in *license.h* begin with RLM_

All the RLM Embedded API functions are described (in alphabetical order) on the following pages.

Table of Contents

rlm_activate() - Request a license activation from the internet

```
#include "license.h"
RLM_HANDLE rh;
int stat;
const char *akey;
int count;
char license[3*(RLM_MAX_LINE+1)];
RLM_ACT_HANDLE act_handle;

rh = rlm_init(...);
stat = rlm_activate(rh, url, akey, count, license, act_handle);
```

rlm_activate() is the preferred call to request license activation from the internet. *url* is the location of the activation server (without the trailing /cgi-bin/ISV_mklic) The activation key *akey* and license count *count* are sent to the server. Note that if the license count is <= 0, a count of 1 is used. Beginning in RLM v9.0, a count of 0 has a special meaning for *NORMAL* activation fulfillments of *floating* licenses – a count of 0 requests that all remaining licenses be fulfilled for this request. In this way, your activation code does not need to supply the number of licenses ordered during fulfillment time. Note that a request of 0 will not retrieve an already-activated license - in order to re-retrieve an already-activated license, you must specify the number of licenses actually generated. If the *akey* parameter contains an embedded newline, *rlm_act_request()* will return RLM_EL_BADPARAM.

Beginning in RLM v9.4, the ISVNAME_ACT_URL environment variable will override the ***url*** parameter to this call. This allows you to change the URL of the activation server without re-building your software. For example, if your ISV name is “demo”, the environment variable would be named “DEMO_ACT_URL”, and you would set it to the URL to use for activation if the ***url*** parameter in this call is no longer correct.

> Note that the URL should ***always*** be http, ***never*** https. *rlm_activate()* encrypts the request independent of the webserver.

Prior to v11.0, rlm would only activate licenses with rehostable, non-zero RLM_HOSTID_32BIT, RLM_HOSTID_ETHER, RLMIDn, RLM_DISKSN, or ISV-defined hostids. Any other hostid will return an RLM_ACT_BAD_HOSTID_TYPE status from *rlm_activate()*. (Note: ISV-defined hostids were added to the list of legal hostids in RLM v4.0). Beginning in v11.0, you can specify exactly the hostids you will accept with the *rlm_isv_cfg_actpro_allowed_hostids()* call in rlm_isv_config.c, then either re-building your license generator or creating a new generator settings file. See Customizing RLM Embedded with rlm_isv_config on page 25 for more details.

The priority is (assuming the particular hostid type is enabled):

- rehostable hostid
- ISV-defined hostid
- rlmid hostid
- Disk Serial Number
- ethernet address
- 32-bit hostid
- ip address hostid
- user-based hostid
- host-based hostid
- serial number hostid

- string hostid
- DEMO hostid
- ANY hostid

If none of the hostid types above are present (or enabled), the activation software will return RLM_ACT_BAD_HOSTID_TYPE. Beginning in RLM v11.2, if RLM_ACT_BAD_HOSTID_TYPE is returned, the "license" parameter will contain the decimal representation of the list of valid hostids (as defined in license.h, in the RLM_ACTPRO_ALLOW_xxx definitions). This parameter is a string which represents a decimal number containing a bitwise OR of the allowed hostid types. To decode the alllowed hostid types from the license string, use code similar to this:

```
allowed = atoi(license);
```

You can override RLM's notion of the hostid by calling *rlm_act_set_handle()* with the RLM_ACT_HANDLE_HOSTID_LIST parameter. The *hostid_list* parameter can contain a list of hostids for use in nodelocked licenses. This is specified with the following syntax:

```
list:list-of-hostids
```

For example:

```
list:user=joe host=sam ip=192.16.7.23 3f902d8b0027
```

If a list is supplied, note the following:

- The activation software uses the hostids in the list as you specified, even if they are not "secure".
- If the license to be activated is a served license (floating), only the first hostid in the list is used.
- The number of available activations on the activation key is decremented by 1 regardless of the number of hostids in the license created.
- The hostid list must be less than RLM_ACT_MAX_HOSTID_LIST characters long (205) including the "list:" prefix.
- The hostid list can contain no more than RLM_MAX_HOSTID_LIST (25) hostids.

  This capability can be used to create a license which works on 2 (or more) systems, e.g. to create a license for a primary and a backup system. It can also be used to pass a hostid of a less secure type to be used, e.g. the *hostid-list* "list:ip=172.16.7.12" will cause the activation software to use the IP address as a hostid without returning RLM_ACT_BAD_HOSTID_TYPE.

If *act_handle* is NULL, no optional parameters are specified. If *act_handle* is passed to *rlm_activate()* as a non-NULL handle, other, lesser-used parameters can be specified:

- isvname – if different from your ISV name.
- hostid – if you do not want to use the default hostid.
- hostname – if you want to change the notion of your hostname
- extra – any extra license parameters

- log – information to log to the activation server (Activation Pro only).

These other parameters are passed in by calling *rlm_act_new_handle()* and *rlm_act_set_handle()*

The parameter *license* must be an allocated string of length 3*(RLM_MAX_LINE+1). If *rlm_activate()* succeeds, the activated license is returned in this string. For certain errors, the *license* string will contain MySQL error information, otherwise it will be an empty string.

Status returns >=0 indicate success, < 0 are failure status.

Status	Meaning
0	license was activated, first request, activation count consumed
1	license previously activated. Activation count is not consumed; the prior license is returned. This status indicates that a duplicate activation key/count/hostid was sent to the server.
RLM_ACT_BADPARAM	Bad parameter to activation function
RLM_ACT_NO_KEY	No Activation key supplied
RLM_ACT_NO_PROD	No product definition (internal database error)
RLM_ACT_CANT_WRITE_KEYS	Cannot write activation keys (admin tool)
RLM_ACT_KEY_USED	Activation key used already (no count remaining)
RLM_ACT_BAD_HOSTID	Missing hostid
RLM_ACT_BAD_HOSTID_TYPE	Invalid hostid type
RLM_ACT_BAD_HTTP	Bad HTTP transaction
RLM_ACT_CANTLOCK	Cannot lock activation database
RLM_ACT_CANTREAD_DB	Cannot read activation database
RLM_ACT_CANT_WRITE_FULFILL	Cannot write fulfillment (licf) table
RLM_ACT_CLIENT_TIME_BAD	Time difference too great from server->client system
RLM_ACT_BAD_REDIRECT	Bad http Redirect
RLM_ACT_TOOMANY_HOSTID_CHANGES	Too many hostid changes for redirect
RLM_ACT_BLACLISTED	Domain on blacklist
RLM_ACT_NOT_WHITELISTED	Domain not on whitelist
RLM_ACT_KEY_EXPIRED	Activation Key expired
RLM_ACT_NO_PERMISSION	HTTP request denied
RLM_ACT_SERVER_ERROR	HTTP internal server error
RLM_ACT_BAD_GENERATOR	Bad or missing license generator file
RLM_ACT_NO_KEY_MATCH	No matching activation key found in database
RLM_ACT_NO_AUTH_SUPPLIED	No proxy authentication credentials supplied
RLM_ACT_PROXY_AUTH_FAILED	Proxy authentication failed
RLM_ACT_NO_BASIC_AUTH	Activation supports only BASIC proxy authentication
RLM_EH_CANTCONNECT_URL	Cannot connect to specified URL

RLM_ACT_GEN_UNLICENSED	Activation generator unlicensed
RLM_ACT_DB_READERR	Activtion DB read error (MySQL)
RLM_ACT_GEN_PARAM_ERR	Generating license - bad parameter
RLM_ACT_UNSUPPORTED_CMD	Unsupported command to generator

If you are using Activation Pro, you should consult the Activation Pro manual for troubleshooting tips and additional error returns.

Proxy Server Support

RLM activation has support for proxy servers. To use a proxy server, there are 2 environment variables which must be set:

HTTP_PROXY- set to the **hostname:port** of the proxy server. For example, if your proxy server is on port **8080** on host **proxy_host**:

```
% setenv HTTP_PROXY proxy_host:8080
```

If your proxy server uses authentication, you can use the HTTP_PROXY_CREDENTIALS environment variable to pass the credentials to the proxy server:

HTTP_PROXY_CREDENTIALS - the username and password to authenticate you to the proxy server, in the format **user:password**. For example, if your username is "joe" and password is "joes_password":

```
% setenv HTTP_PROXY_CREDENTIALS joe:joes_password
```

Note that RLM activation supports only the BASIC authentication type.

You can either set these environment variables before running your application, or use putenv() (or rlm_putenv()) to set them inside your application before calling *rlm_activate()*.

rlm_activate() encrypts the data sent to the activation server. Beginning in RLM v9.1, if RLM_ACT_NO_ENCRYPT is set in the environment, *rlm_act_request()* will not encrypt the data sent to the activation server.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_act_info() - Get info about an activation key from the server.

```
#include "license.h"
RLM_HANDLE rh;
const char *url = “your activation server URL here”;
char *akey = “activation-key-desired”;
char product[RLM_MAX_PRODUCT+1];
char version[RLM_MAX_VER+1];
char upgrade_version[RLM_MAX_VER+1];
int  date_based;
int license_type;
int status;

rh = rlm_init(...);
status = rlm_act_info(rh, url, akey, product, version, &date_based, &license_type,
upgrade_version);
```

The *rlm_act_info()* call presents the activation key ***akey*** to the server at ***url*** and retrieves information about the license which would be generated by this key.

> Note that the URL should ***always*** be http, ***never*** https. *rlm_act_info()* encrypts the request independent of the webserver.

NOTE: ***Prior to RLM v11.0, rlm_act_info() returned the information for disabled activation keys. Beginning in RLM v11.0, rlm_act_info() will return RLM_ACT_KEY_DISABLED with no further information for disabled activation keys.***

The rlm_act_info() call returns 0 for success, or an RLM errror code otherwise.

The returned information is passed back in the last 5 parameters:

- product – the product name in the license that would be generated from this activation key.
- version – the version in the generated license. If date_based is non-zero, this is a string representing an integer number of months; the version is a date-based version of the form yyyy.mm for this number of months after license generation. If date_based is 0, the actual license version is returned in this parameter.
- date_based – non-zero indicates that the version string is the number of months after license generation for a date-based version.
- license_type – this is the type of license that will be generated. These types are defined in license.h:

  ```
  #define RLM_ACT_LT_FLOATING 0   /* Floating */
  #define RLM_ACT_LT_F_UPGRADE 4  /* Floating UPGRADE */
  #define RLM_ACT_LT_UNCOUNTED 1  /* Nodelocked, Uncounted */
  #define RLM_ACT_LT_NLU_UPGRADE 5 /* Nodelocked, Uncounted UPGRADE */
  #define RLM_ACT_LT_SINGLE 3     /* Single */
  ```

#define RLM_ACT_LT_S_UPGRADE 7 /* Single UPGRADE */

- upgrade_version – the version eligible for an upgrade for UPGRADE type licenses. This is always a fixed string (ie, it is never date-based). For non-upgrade licenses, this will be an empty string.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_act_keyinfo2() - Get the most info about an activation key

rlm_act_fulfill_info() - Get info about key and latest fulfillment.

rlm_act_info() - Get info about an activation key from the server.

rlm_act_keyinfo() - Get info about an activation key from the server.

```
#include "license.h"
RLM_HANDLE rh;
const char *url = “your activation server URL here”;
char *akey = “activation-key-desired”;
char product[RLM_MAX_PRODUCT+1];
char version[RLM_MAX_VER+1];
char upgrade_version[RLM_MAX_VER+1];
char exp[RLM_MAX_EXP+1];
char keyexp[RLM_MAX_EXP+1];
char hostid[RLM_MAX_HOSTID_STRING+1];
int  date_based;
int license_type;
int status;
int count, fulfilled, rehosts, revoked, allowed_hostids, sub_interval, sub_window;

rh = rlm_init(...);
```

status = ***rlm_act_keyinfo2*(rh, url, akey, product, version, &date_based, &license_type, upgrade_version, &count, &fulfilled, &rehosts, &revoked, exp, hostid, keyexp, &allowed_hostids, &sub_interval, &sub_window);**

(note: rlm_act_keyinfo2() returns a superset of the information from all the other calls, and it is the preferred call).

status = ***rlm_act_info*(rh, url, akey, product, version, &date_based, &license_type, upgrade_version);**

status = ***rlm_act_keyinfo*(rh, url, akey, product, version, &date_based, &license_type, upgrade_version, &count, &fulfilled, &rehosts, &revoked);**

status = ***rlm_act_fulfill_info*(rh, url, akey, product, version, &date_based, &license_type, upgrade_version, &count, &fulfilled, &rehosts, &revoked, exp, hostid);**

The *rlm_act_info()* call presents the activation key ***akey*** to the server at ***url*** and retrieves information about the license which would be generated by this key.

The *rlm_act_keyinfo()* returns everything that *rlm_act_info()* returns, plus some fulfillment information about the activation key.

The *rlm_act_fulfill_info()* returns everything that *rlm_act_keyinfo()* returns, plus the actual expiration date and hostid from the most recent fulfillment on the activation key. If no fulfillments have been made (i.e. fulfill == 0), the return values exp and hostid are undefined.

> Note that the URL should ***always*** be http, ***never*** https. *rlm_act_info()* encrypts the request independent of the webserver.

NOTE: ***Prior to RLM v11.0, rlm_act_info() returned the information for disabled activation keys. Beginning in RLM v11.0, rlm_act_info() will return RLM_ACT_KEY_DISABLED with no further information for disabled activation keys.***

The rlm_act_info() call returns 0 for success, or an RLM errror code otherwise.

The returned information is passed back in the last 5 parameters:

- product – the product name in the license that would be generated from this activation key.
- version – the version in the generated license. If date_based is non-zero, this is a string representing an integer number of months; the version is a date-based version of the form yyyy.mm for this number of months after license generation. If date_based is 0, the actual license version is returned in this parameter.
- date_based – non-zero indicates that the version string is the number of months after license generation for a date-based version.
- license_type – this is the type of license that will be generated. These types are defined in license.h:

  #define RLM_ACT_LT_FLOATING 0 /* Floating */

  #define RLM_ACT_LT_F_UPGRADE 4 /* Floating UPGRADE */

  #define RLM_ACT_LT_UNCOUNTED 1 /* Nodelocked, Uncounted */

  #define RLM_ACT_LT_NLU_UPGRADE 5 /* Nodelocked, Uncounted UPGRADE */

  #define RLM_ACT_LT_SINGLE 3 /* Single */

  #define RLM_ACT_LT_S_UPGRADE 7 /* Single UPGRADE */
- upgrade_version – the version eligible for an upgrade for UPGRADE type licenses. This is always a fixed string (ie, it is never date-based). For non-upgrade licenses, this will be an empty string.

In addition to the above information, *rlm_act_keyinfo()* returns fulfillment information:

- count – the allowed fulfillment count (0 = unlimited)
- fulfilled – the # already fulfilled
- rehosts – the number of rehost operations allowed
- revoked – the number of revocations already performed

  Note that when revoked==rehosts, no additional license revocations will be allowed.

In addition to the above information, *rlm_act_fulfill_info()* returns recent fulfillment information:

- exp – the actual expiration date of the latest fulfillment.
- hostid – the hostid from the latest fulfillment.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_act_keyvalid(), rlm_act_keyvalid_license() - Verify that an activation key still has a valid license on this hostid from the activation server.

```
#include "license.h"
RLM_HANDLE rh;
const char *url = “your activation server URL here”;
char *akey = “activation-key-desired”;
char hostid[RLM_MAX_HOSTID+1];
char license[RLM_ACT_MAX_LICENSE+1];
int status;

rh = rlm_init(...);
```
status = ***rlm_act_keyvalid*(rh, url, akey, hostid);**
status = ***rlm_act_keyvalid_license*(rh, url, akey, hostid, license);**

The *rlm_act_keyvalid()* call presents the activation key ***akey*** and ***hostid*** to the server at ***url*** and retrieves status of fulfilled licenses on this hostid for this activation key.

> Note that the URL should ***always*** be http, ***never*** https. *rlm_act_keyvalid()* encrypts the request independent of the webserver.

The *rlm_act_keyvalid()* call returns:

- 0 for success, ie, a non-revoked license has been generated on this hostid for this activation key.
- RLM_ACT_KEY_DISABLED if the activation key itself is disabled.
- RLM_ACT_KEY_NO_HOSTID if there is no fulfilled license matching this hostid for this activation key, or
- RLM_ACT_KEY_HOSTID_REVOKED if the only fulfilled license(s) for this hostid on this activation key have been revoked, or
- RLM_EH_ACT_OLDSERVER or RLM_ACT_UNSUPPORTED_CMD if the activation server is too old to process this request.

There is no other returned information.

The *rlm_act_keyvalid_licenese()* call performs the same operation with the same return as rlm_act_keyvalid(), but in addition, it returns the license if the status return is 0. Note that in the case of a floating license which has had multiple fulfillments, the license returned will be one of the licenses generated with this activation key (in general, the first license generated). If the status return is non-zero, the contents of *license* are undefined. *rlm_act_keyvalid_license()* is new in RLM v11.1, and requires an RLM v11.1 activation pro server to return the license.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_act_new_handle(), rlm_act_destroy_handle() - Create/destroy handle to pass activation parameters.

```
#include "license.h"
RLM_HANDLE rh;
RLM_ACT_HANDLE act_handle;

rh = rlm_init(...);
act_handle = rlm_act_new_handle(rh);

(void) rlm_act_destroy_handle(act_handle);
```

rlm_act_new_handle() creates a blank handle to pass optional activation parameters to *rlm_activate()*. *rlm_act_new_handle()* returns a NULL handle on error.

Call *rlm_act_new_handle()* before calling *rlm_act_set_handle()*. After activation is complete, call *rlm_act_destroy_handle()* to free the memory associated with the handle.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_act_set_handle() - Set data in activation handle.

```
#include "license.h"
RLM_ACT_HANDLE act_handle;
int stat;
int what;
void *val;

stat = rlm_act_set_handle(act_handle, what, val);
```

rlm_act_set_handle() sets various options in an activation handle in order to pass these activation parameters to *rlm_activate()*. *rlm_act_new_handle()* returns 0 for success, RLM_EH_BADPARAM on error (no handle supplied, no value supplied, or bad "what" value).

Call *rlm_act_set_handle()* after calling *rlm_act_new_handle()*. After activation is complete, call *rlm_act_destroy_handle()* to free the memory allocated in the handle.

"what" values:

All values for the "what" parameter are defined in license.h:

- **RLM_ACT_HANDLE_DISCONN**: (int val)

  If set to 1, rlm_activate() will create a rehostable hostid, then return data in the "license" parameter. Take this data to an internet-connected system, pass the data back into rlm_activate() in the hostid_list parameter when requesting the activation (again with RM_ACT_HANDLE_DISCONN set). See the "License Rehosting" appendix in the Activation Pro manual for details on how to make these calls.

- **RLM_ACT_HANDLE_EXTRA**: (const char * val)

  This parameter is used to pass extra license options to the activation server. v*al* is a string containing extra "keyword=value" license attributes. These must be valid RLM license syntax, not just any keyword=value pair. Note that the *val* string should not contain characters illegal in license files, and most particularly, it should not contain the '&' character, which is illegal in a license file and also is the cgi separator in web requests. If you put space-separated strings into the *extra* parameter, be sure to enclose them in quotes. For example: set extra to "customer=\"Your Customer Name Here\"" in order to put your customer name into the generated license, or set it to "customer=\"Your Customer Name Here\" min_timeout=100" to set your customer name and the minimum timeout.

- **RLM_ACT_HANDLE_HOSTID_LIST**: (const char * val)

  This parameter is used if you want to pass a particular hostid (other than the default) or a list of hostids to the activation server.

  the *hostid_list* parameter can contain a list of hostids for use in nodelocked licenses. This is specified with the following syntax:

  ```
  list:list-of-hostids
  ```

  For example:

```
list:user=joe host=sam ip=192.16.7.23 3f902d8b0027
```

If a list is supplied, note the following:

- The activation software uses the hostids in the list as you specified, even if they are not "secure".
- If the license to be activated is a served license (floating), only the first hostid in the list is used.
- The number of available activations on the activation key is decremented by 1 regardless of the number of hostids in the license created.
- The list will not be accepted by the server if encryption of the request is turned off with RLM_ACT_NO_ENCRYPT

This capability can be used to create a license which works on 2 (or more) systems, e.g. to create a license for a primary and a backup system. It can also be used to pass a hostid of a less secure type to be used, e.g. the *hostid-list* "list:ip=172.16.7.12" will cause the activation software to use the IP address as a hostid without returning RLM_ACT_BAD_HOSTID_TYPE.

- **RLM_ACT_HANDLE_HOSTNAME**: (const char * val)

  This parameter is used for the (rare) case where you want to pass a specific hostname to the activation server.

- **RLM_ACT_HANDLE_LOG**: (const char * val)

  This parameter is used to pass a string to be logged in the activation server database. This parameter will override any setting of the RLM_ACT_LOG environment variable. *(Note: the use of the RLM_ACT_LOG environment variable is deprecated, and is not guaranteed to work in all future versions of RLM. Setting logging using the rlm_act_set_handle() call is preferred).*

  RLM Activation Pro allows you to log an arbitrary string to the database every time you fulfill a license. This string can be up to 80 characters in length and it will appear in the 'log' column in the ***licf*** table.

- **RLM_ACT_HANDLE_ISV**: (const char * val)

  This parameter, which takes a (char *) value, is used to set the ISVname, if it is different from your ISV name. This will not normally be used. It is used, for example, in the rlm web interface to request an activation from a specified ISV's activation server.

- **RLM_ACT_HANDLE_PRODUCT**: (char * val)

  This parameter, which takes a (char *) value, is used to set the product name when you are preparing to do activation of a rehostable hostid on a disconnected system. This will not normally be used. RLM_ACT_HANDLE_DISCONN should also be set when the product name is set. See the Activation Pro manual for more information.

- **RLM_ACT_HANDLE_REHOST**: (int val)

  If set to 1, rlm_activate() will create a rehostable hostid, then activate the license using that rehostable hostid. If the hostid already exists for the product associated with the activation key, rlm_activate() will return RLM_EH_REHOST_EXISTS and will not proceed with the activation Once created, the contents of the rehostable hostid directory ***CANNOT BE TOUCHED, MODIFIED, DELETED, or RESTORED from a BACKUP*** *without invalidating the hostid. NOTE: REHOSTable hostids can be used with nodelocked, uncounted, and SINGLE licenses only.*

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_act_rehost_revoke() - revoke a rehostable license

rlm_act_revoke_disconn() - revoke a disconnected rehostable license

rlm_act_revoke() - revoke a rehostable license *(depricated)*

rlm_act_revoke_reference() - revoke a rehostable hostid by "reference" hostid. *(depricated)*

```
#include "license.h"
RLM_HANDLE rh;
char *url;
char *product, *param;
char retval[RLM_ACT_MAX_LICENSE+1];
int stat, flags;

rh = rlm_init(...);
stat = rlm_act_rehost_revoke(rh, url, product, flags);
stat = rlm_act_revoke_disconn(rh, url, param, retval);
stat = rlm_act_revoke(rh, url, product); (depricated)
stat = rlm_act_revoke_reference(rh, url, product);  (depricated)
```

rlm_act_rehost_revoke() causes RLM to revoke a rehostable hostid by taking the following actions:

- contacts the activation server at *url* and tells it to revoke all activations performed for the revokable hostid for product *product*.
- removes the hostid for *product* from the system

flags is one or more of the following values, ORed together:

- RLM_ACT_REVOKE_REFERENCE – revoke using the reference hostid if the hostid is not found on this system.
- RLM_ACT_REVOKE_NOFULFILL – removes the rehostable hostid even if the activation server does not think there is a fulfillment.
- RLM_ACT_REVOKE_NODEL – tells the server to revoke the fulfillment even if the rehostable hostid cannot be deleted.

If *rlm_act_rehost_revoke()* cannot contact the activation server, or no fulfillments have been made using rehostable hostids for *product*, or the rehostable hostid for *product* does not exist, amd none of the 3 possible conditions specified in *flags* corrects this, *rlm_act_rehost_revoke()* will return a non-zero error status. Otherwise, *rlm_act_rehost_revoke()* returns a 0 status to indicate success.

> Note that the URL should ***always*** be http, ***never*** https. *rlm_act_revoke()* encrypts the request independent of the webserver.

If you set a reference hostid with rlm_set_attr_reference_hostid() prior to activating this license, be sure to set the same reference hostid before attempting to revoke the license.

Once a license is revoked with *rlm_act_rehost_revoke()*, it will no longer work on the system, and activation count associated with the fulfillment to this system will be returned to the activation server so that your customer can re-activate the license on another system.

See the notes below for considerations when setting the various flags bits.

rlm_act_revoke_disconn() is used to perform a rehostable hostid revocation on a system which is not connected to the internet. For the usage of this function, see the "License Rehosting" appendix in the RLM Activation Pro manual.

Note/risks when setting the *flags* bits to *rlm_act_rehost_revoke()*:

- *RLM_ACT_REVOKE_REFERENCE* – this should always be a safe option to set, unless you are worried that your customer has the same native hostid for multiple systems. In that case, the license could be revoked on a different system, and the original system would still have a valid rehostable hostid. If you already call *rlm_revoke_reference()* after *rlm_revoke()* fails, you should set this option and not worry about it.
- *RLM_ACT_REVOKE_NOFULFILL* – in this case, the server did not find the fulfillment, but with this option set, the rehostable will be deleted. This prevents a new activation from receiving a "rehostable hostid exists" error, and is a good option to set.
- *RLM_ACT_REVOKE_NODEL* – this option tells the server to remove the fulfillment even if the rehostable cannot be deleted. This would allow a dishonest customer to cheat the system by first write-protecting the rehostable directory hierarchy, then revoking the license. With this option set, even if the hostid can't be deleted, *rlm_act_rehost_revoke()* would still tell the activation server that it was deleted, and the server would put activation count back into the key.

  Reprise Software recommends always setting RLM_ACT_REVOKE_REFERENCE and RLM_ACT_REVOKE_NOFULFILL, and adding RLM_ACT_REVOKE_NODEL depending on your tolerance for some customers being able to cheat the system.

Notes on the deprecated calls

rlm_act_revoke() - the pre-14.1 revoking API call, is still available, and it is 100% equivalent to calling *rlm_act_rehost_revoke(rh, url, product, 0)*. New applications should call *rlm_act_rehost_revoke()* with the appropriate *flags*.

There is no longer any reason to call *rlm_act_revoke_reference()* after a failure of *rlm_act_revoke()*. *rlm_act_rehost_revoke(rh, url, product, RLM_ACT_REVOKE_REFERENCE)* is equivalent to *rlm_act_revoke()* followed by *rlm_act_revoke_reference()*.

rlm_act_revoke_reference() performs the same operation as *rlm_act_revoke()*, but it will work even when the rehostable hostid is bad or missing on the system. You must decide if you are willing to revoke the license in this case, and you should only call rlm_act_revoke_reference() after rlm_act_revoke() fails with an RLM_EH_CANT_GET_REHOST or RLM_EL_NOTTHISHOST status.

Note that rlm_act_rehost_revoke() and rlm_act_revoke() will return RLM_ACT_REVOKE_TOOLATE (-1029) if the license associated with the rehostable hostid has expired, and you have not enabled "Revocation of expired rehostable hostids" in the Database section of the Admin tab in RLM License Center. This error means that no count was returned to the activation key, however, the rehostable hostid was deleted in this case. If there is sufficient count in the activation key, or if a different activation key is used, a new rehostable activation will succeed.

The most likely scenario where you would see this is as follows:

1. **user attempts to check out license, gets RLM_EL_EXPIRED status.**
2. **user then attempts to re-activate the license, gets RLM_EH_REHOST_EXISTS**
3. **user then attempts to revoke the activation, gets RLM_ACT_REVOKE_TOOLATE**

at this point, the rehostable hostid is gone, and the user can re-activate successfully.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_add_isv_hostid() - Enable ISV custom hostid processing

```
#include "license.h"
RLM_HANDLE rh;
char *keyword;
int type;
int transient;
int  (*func(char *, RLM_HANDLE));

status = rlm_add_isv_hostid(rh, keyword,  type,  transient, int (*func)())
```

where:

rh - is the RLM_HANDLE
keyword - is the hostid's keyword as described below
type - is the hostid's type as described below
transient - is set to a non-zero value if the hostid can change
func - is the hostid's runtime discovery function as described below

rlm_add_isv_hostid() returns:

0	if successful
RLM_EH_DUP_ISV_HID	if the keyword and/or type has already been used in another ISV-defined hostid
RLM_EH_BADPARAM	if the keyword is NULL or its length is > 10 bytes, or if type is < RLM_ISV_HID_TYPE_MIN

Certain ISVs require their own private hostid type, especially to support an inventory of existing dongles. This section describes how to support your own hostid in RLM.

Note: Reprise Software discourages the use of ISV-defined hostids. If you use ISV-defined hostids:

- **You cannot use a ISV server settings file, instead, you must ship a binary, which means your customers cannot get bug fixes with a new generic rlm server.**
- **Your customers cannot use the generic rlmhostid tool, you must write and ship your own tool.**
- **You cannot use the standard Activation Pro license generator, you must build a custom generator - which may involve licensing an additional RLM platform.**

An ISV-defined hostid consists of 3 parts:

1.A ***keyword*** which identifies the hostid type in a license (eg, hostid=MYHOSTID=abcdefghi). MYHOSTID is the keyword, which you chose. The keyword must be <= 10 characters long, and must be unique among all your hostid types.

2.An integer ***type***, which is used by RLM to differentiate among hostid types. The *type* of a hostid must be >= RLM_ISV_HID_TYPE_MIN, and must be unique across all your hostid types. You choose this number.

3. A ***function*** you write which is used to determine the value of the hostid of that type at runtime. For instance if the ISV-defined hostid type was a dongle ID, the function would read the dongle's ID and return that value. The function's prototype is:

int <name>(char *, RLM_HANDLE)

RLM calls this function at server runtime and/or licensed application runtime. The function should return 0 if it succeeds, or non-zero if it doesn't. The function returns the hostid as a NULL-terminated string in the char array pointed to by the function argument. The returned string must not exceed 64 bytes in length (RLM_MAX_HOSTID), and may not contain any white space, and must consist of all printable characters. The string space is allocated on the stack (RLM_MAX_HOSTID+1 bytes) and initalized to all 0's, so if you write more than RLM_MAX_HOSTID bytes, you risk corrupting data inside your application or license server and/or crashing the process. Your function will be passed an empty RLM_HANDLE as it's second parameter. You can use this handle if you need to call RLM functions which require a handle. This parameter is new in RLM v9.1.

Note that beginning in RLM v8.0, the three conditions on ISV-defined hostid strings are enforced. These conditions are:

- must be <= 64 characters
- can contain no whitespace, and
- must consist of all printable characters.

These restrictions were not enforced prior to RLM v8.0. If any of these conditions are violated, the hostid will not be processed, and will produce an RLM_HOSTID_INVALID hostid type.

You may have zero, one, or more ISV-defined hostids. Each hostid must have a unique type, keyword, and function. Each ISV-defined hostid type is identified to RLM via a function call made to *rlm_add_isv_hostid()* in your rlm_isv_config.c module.

Note that *rlm_add_isv_hostid()* must be called from rlm_isv_config.c for each of your ISV-defined hostid types. See the example in the rlm_isv_config.c file shipped on the kit. Do not call *rlm_add_isv_hostid()* from anywhere else in your application.

Transient hostids

A hostid can be *transient* or not. A *transient* hostid is one which may change state, and needs to be checked periodically. For example, a dongle would be considered a *transient* hostid, because it can be removed from the computer while the software is running. If your hostid type is *transient*, be sure to set the *transient* parameter of this call to a non-zero value. If the hostid type is *transient*, the hostid will be verified once per minute in the license server and each time that *rlm_get_attr_health()* is called in your application (but not more often than once every 30 seconds, unless the hostid does not return the correct value, in which case it is checked each time *rlm_get_attr_health()* is called).

Supporting multiple instances of a single hostid type

In rare circumstances, it is necessary to support multiple instances of a single hostid type. This may be the case, for example, if you need to support multiple dongleIDs on a single machine.

For this case, there is an alternate call, ***rlm_add_isv_hostid_multiple()***. This call takes an extra function parameter, as described in the next section.

Client-side hostid processing

When an application requests a license from a license server, it will transmit the hostid information from the local machine to the license server, so that the server can process node-locked licenses without additional queries to the application. The application will transmit a maximum of 25 different hostids:

- one 32-bit hostid, if present on this platform
- one disk hardware serial number (disksn=) - Windows only
- one IP address
- up to 8 ethernet MAC addresses (ether=)
- a minimum of 3 ISV-defined hostids (usually more, but guaranteed to be at least 3)

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_add_isv_hostid_compare() - Install ISV custom hostid comparison routine

status = ***rlm_add_isv_hostid_compare(RLM_HANDLE rh, int type, int (*compare)(int, char *, char *))***

where:

rh is the RLM_HANDLE
type is the hostid's type
compare() is the comparison routine

RLM, by default, compares the hostid in the license file with the return from your ISV-defined hostid routine using a strcasecmp() function. However, in some cases, it is desirable to have another comparison routine, for example, if you want to allow for some differences between the hostid on the current system and what is in the license. In order to accomplish this, you can register a comparison routine for any ISV-defined hostid you have registered. To do this, call *rlm_add_hostid_compare().* Once installed, when RLM needs to compare 2 instances of the ISV-defined hostid ***type***, it will call the compare*()* routine to do this comparison rather than the default algorithm (strcasecmp()).

compare() is called as follows:

status = **compare*(int type, char *hostid1, char *hostid2);

where:

type is the hostid's type
hostid1 and hostid2 are the 2 hostids to be compared.

compare() should return 0 if the hostids are the same, and RLM_EL_NOTTHISHOST if the two hostids are not the same.

Note that you cannot assume that you are running on the system which produced *either hostid*. You will be running on a different machine, for example, when the license server is verifying client hostids for node-locked licenses which it is serving. This machine may be a different platform type as well. Also note that if you are using hostid comparison routines, you must register your ISV-defined hostid and comparison routine in your ISV server, even if you only issue nodelocked licenses using your ISV-defined hostid. This is because the server must be able to execute your comparison routine.

The comparison routine is passed 2 hostids. It is undefined which hostid came from the license file and which came from the running system. In fact, this routine will be called on occasion to compare hostids from 2 different license files, in which case neither hostid came from a client system.

rlm_add_isv_hostid_multiple() returns:

0 if successful, RLM_EH_BADPARAM if the hostid type is not already defined.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_add_isv_hostid_multiple() - Enable ISV custom hostid processing - support multiple instances

status = ***rlm_add_isv_hostid_multiple(RLM_HANDLE rh, char *keyword, int type, int transient, int (*func_first)()), int (*func_next)())***

where:

rh is the RLM_HANDLE
keyword is the hostid's keyword
type is the hostid's type
transient is set to a non-zero value if the hostid can change
func_first returns the first instance of the hostid - same as the *func* argument to **rlm_add_isv_hostid()**
func_next returns the **next** instance of the hostid

RLM will call the *func_first()* routine first. You should initialize the list of hostid instances in this function, and return the first one, then prepare to return each additional one in turn when *func_next()* is called. Return 0 if there is at least one instance of this hostid type, and non-zero if no instances are present on this machine.

RLM will then call the *func_next()* routine until it returns a non-zero result. You should return each additional hostid instance in turn when *func_next()* is called. Return 0 if *func_next()* is returning a hostid instance, and non-zero if no more instances are present on this machine. NOTE: *func_next()* may be called even if *func_first()* returns a non-zero value; in this case, *func_next()* should return a non-zero value as well.

rlm_add_isv_hostid_multiple() returns:

0 if successful
RLM_EH_DUP_ISV_HID if the keyword and/or type has already been used in another ISV-defined hostid
RLM_EH_BADPARAM if the keyword is NULL or its length is > 10 bytes, or if type is < RLM_ISV_HID_TYPE_MIN

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_checkin() - Release a license

```
#include "license.h"
RLM_LICENSE license;

rlm_checkin(license);
```

rlm_checkin() releases *license* and frees all data associated with it. After calling *rlm_checkin()*, the RLM_LICENSE *license* is no longer valid, and you should make no further calls using this handle. Do not call *rlm_checkin()* more than once on a license.

Note: you cannot call *rlm_license_stat()* on a license handle after that handle has been checked in, or if the RLM_HANDLE used to check it out has been closed. In fact, you cannot use this handle in any way. Use of the handle after an *rlm_checkin()* or *rlm_close()* will result in unpredictable behavior (including possible application crashes), since the handle you are using has been freed by RLM.

Also Note: If you plan to check any licenses in then close the handle (ie, if you are not going to use the handle after checking a license in), then you should omit the r*lm_checkin()* call, and simply call *rlm_close()* on the handle. *rlm_close()* always checks-in any licenses which are checked out on the handle, and if you are using a disconnected handle, RLM will only reconnect to the server one time for all your license checkins as well as to tell the server that your are done with the handle.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_checkout() - Request a license from RLM

```
#include "license.h"
RLM_HANDLE handle;
RLM_LICENSE license;
const char *product;
const char *version;
int count;

license = rlm_checkout(handle, product, version, count);
```

rlm_checkout requests *count* licenses of product *product* at version *version*. *count* must be a positive integer. If there are node-locked, uncounted licenses available in a license file that is specified in The License Environment, then these node-locked licenses are used. Otherwise, a request is made to each server specified by The License Environment, until either the licenses are granted, or all servers have been tried without success. *rlm_checkout()* creates *license* and returns it to its caller. The *version* string should be of the form *major* or *major.minor* where *major* and *minor* are integers. The *count* parameter must be a positive integer.

The order of license checkout attempts is as follows:

- If **RLM_ROAM** is set to a positive value, roamed licenses on the local node will be checked first,
- All node-locked, uncounted licenses in local license files (from all license files in the license file path) will be checked next
- All licenses served by servers that RLM has already connected to are checked next,
- All licenses served by servers which RLM has not previously connected to are checked next,
- Finally, if RLM_ROAM is not set, a check will be made for local roamed licenses.

To get the status of the *rlm_checkout* call, use *rlm_license_stat(license)*. For a list of status returns, see Appendix B – RLM Status Values on page 137.

There are generally 3 "success" status returns from a license checkout request:

- 0 - license checked out normally
- RLM_EL_OVERSOFT - license checkout results in usage over the soft_limit specified, or a token-based license is misconfigured and the server is in an overdraft condition (see note in the token-based license restrictions section).
- RLM_EL_INQUEUE - license request is in the queue.

If you have specified a minimum server version/revision/build via the rlm_isv_cfg_set_oldest_server() call in rlm_isv_config.c, and the server is older than your specification, you will get an RLM_EL_COMM_ERROR error from the server and the handle will have the error status RLM_EH_SERVER_REJECT.

NOTE: You should always call *rlm_checkin()* when you are done with the license, even if the checkout call returns an error. Calling *rlm_checkin()* on the license frees any associated memory with the license. You can call *rlm_checkin()* even if *rlm_checkout()* returns a NULL license handle, however, you should only call *rlm_checkin()* on a non-NULL license handle once.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_checkout_product() - Request an exact license from RLM

```
#include "license.h"
RLM_HANDLE handle;
RLM_LICENSE license;
RLM_PRODUCTS product;
const char *version;
int count;

license = rlm_checkout_product(handle, product, version, count);
```

In most cases, applications use *rlm_checkout()* to check out licenses, as any license that meets the product name and version requirements is sufficient. In some other cases, an application may want to choose among multiple instances of licenses for the same product. For example, if there are several licenses present for a product but they contain different options attributes, the application may want to check out a specific instance based on the options content determined with *rlm_products()*. In that case the application would use *rlm_checkout_product()* to check out the license.

rlm_checkout_product() requests *count* licenses of version *version* of the product specified by the RLM_PRODUCTS handle *product*. *count* must be a positive integer. *rlm_checkout_product()* creates *license* and returns it to its caller. The *version* string should be of the form *major* or *major.minor* where *major* and *minor* are integers. The *count* parameter must be a positive integer.

rlm_checkout_product() operates on the RLM_PRODUCTS handle returned from *rlm_products()*. Once you have found the product you want to check out via the *rlm_product_first()* and *rlm_product_next()* calls, a call to *rlm_checkout_product()* will check out the product that is described by the current state of the RLM_PRODUCTS handle *product*.

To get the status of the *rlm_checkout_product()* call, use *rlm_license_stat(license)*. For a list of status returns, see Appendix B – RLM Status Values on page 137.

There are generally 3 "success" status returns from a license checkout request:

- 0 - license checked out normally
- RLM_EL_OVERSOFT - license checkout results in usage over the soft_limit specified or a token-based license is misconfigured and the server is in an overdraft condition (see note in the token-based license restrictions section).
- RLM_EL_INQUEUE - license request is in the queue.

If you have specified a minimum server version/revision/build via the rlm_isv_cfg_set_oldest_server() call in rlm_isv_config.c, and the server is older than your specification, you will get an RLM_EL_COMM_ERROR error from the server and the handle will have the error status RLM_EH_SERVER_REJECT.

NOTE: You should always call *rlm_checkin()* when you are done with the license, even if the checkout call returns an error. Calling *rlm_checkin()* on the license frees any associated memory with the license. You can call *rlm_checkin()* even if *rlm_checkout()* returns a NULL license handle, however, you should only call *rlm_checkin()* on a non-NULL license handle once.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_close() - Terminate licensing operations with RLM

```
#include "license.h"
RLM_HANDLE handle;

rlm_close(handle);
```

When you are finished with all licenses and do not intend to make any more calls to RLM, call *rlm_close()* to clean up the handle created with *rlm_init()* and free all the data associated with it.

rlm_close() does the following:

- if you have automatic heartbeats - syncs with the other thread and destroys that thread
- checks in any licenses that are still checked out - which will disconnect from all servers and shut down the connections (and on windows, calls WSACleanup() to close down Winsock).
- frees all data structures used in that handle.
- frees the handle.

Note: If you are using a DLL on Windows, you ***cannot*** call *rlm_close()* in the DLL unloading routine.

Note: you cannot use any license handles that were created using this RLM_HANDLE after the call to *rlm_close()*. Use of the RLM_HANDLE or any associated license handles after an *rlm_close()* will result in unpredictable behavior (including possible application crashes), since the handle you are using has been freed by RLM.

Also Note: If you plan to check any licenses in then close the handle (ie, if you are not going to use the handle after checking a license in), then you should omit the r*lm_checkin()* call, and simply call *rlm_close()* on the handle. *rlm_close()* always checks-in any licenses which are checked out on the handle, and if you are using a disconnected handle, RLM will only reconnect to the server one time for all your license checkins as well as to tell the server that your are done with the handle.

You are not strictly required to call *rlm_close()* ***unless the handle is a disconnected handle***. Specifically, if your program is about to exit, *rlm_close()* is unnecessary for a connected handle, but for a disconnected handle, *rlm_close()* informs the server that you are done and allows the server to clean up data associated with your process. Of course, you can omit the rlm_close() call even for a disconnected handle, in which case the server will time out the licenses after your *promise* interval.

If you do not call *rlm_close()*, memory leak detectors will report leaked memory. Also note that there are some idiosyncrasies in the OpenSSL package which can cause memory leaks to be reported. In particular, if you have a license with a bad signature, OpenSSL allocates several hundred bytes of memory that doesn't normally get freed. To free it and keep leak detectors quiet, call

```
ERR_remove_state(0);
```

just before exiting your program. Do *NOT* call this function if you are going to continue using RLM in another handle.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_detached_demo() - Install RLM Detached Demo™ license

```
#include "license.h"
RLM_HANDLE rh;
int stat;
int days;
const char license[RLM_MAX_LINE+1];

rh = rlm_init(...);
stat = rlm_detached_demo(rh, days, license);
```

rlm_detached_demo() requests RLM to install a Detached Demo™ valid for *days*. The parameters of the demo license installed are contained in the *license* string.

days - the number of days the demo license should be valid.
license - an RLM license string.

When installing the demo license, the *license* string is parsed into it's components and these are used for the license. The license should not be signed, but must have valid syntax, otherwise an RLM_EL_BADPARAM error will result. If *rlm_detached_demo()* returns a non-zero status, the status is contained in the RLM_HANDLE parameter (rh) after the call.

The *count*, *hostid*, and *expiration date* fields of this input *license* are unused. The resulting installed demo license will be a node-locked, uncounted license, valid on the machine which made the call to *rlm_detached_demo(),* ***and valid for the version specified in the license only.*** The expiration date will be *days* days in the future. Note that licenses are valid until midnight, local time, so a 0-day license will be valid until midnight on the day it is installed.

A Detached Demo™ license can only be installed once on a particular system for any given combination of *product* and *version*. Detached Demo™ licenses cannot be modified or re-installed. They do not require any kind of internet connectivity, however, they are not as secure as licenses created with RLM Activation Pro, which is always the preferred way to install a license which expires in a fixed number of days.

For a Detached Demo™ license to be usable, you must be able to check out an rlm_demo license. This allows you to add the code to create demo licenses into your product, but enable it only in certain situations. If you call *rlm_detached_demo()* without an rlm_demo license available, the operation will fail with an RLM_EH_NO_DEMO_LIC status. Note that the rlm_demo license must be valid, in other words, you must sign this license and it must be present *and valid* on the system where the demo is going to be installed. The rlm_demo license should be placed in the directory with your product binary, and it should be a nodelocked, uncounted license, perhaps locked to hostid *demo* or *any*, e.g.:

LICENSE demo rlm_demo 1.0 permanent uncounted hostid=demo

The following example is a call to *rlm_detached_demo()* to set up a 30-day license for v1.0 of *myproduct*:

```
RLM_HANDLE rh;
int stat;
char license[RLM_MAX_LINE+1];
```

```
rh = rlm_init(...);
sprintf(license, "LICENSE demo mylicense 1.0 permanent uncounted hostid=any _customer=%s",
customer);
stat = rlm_detached_demo(rh, 30, license);
```

To determine if a license which is checked out is a Detached Demo[tm] license, call *rlm_license_detached_demo()* on the license handle. If it is a Detached Demo[tm] license, *rlm_license_detached_demo()* will return 1.

Note: Detached Demo[tm] licenses are not as secure as licenses created with RLM Activation Pro. Using internet activation to install demo licenses is always preferred, and Detached Demo[tm] licenses should only be used when absolutely required. Also note that Detached Demo[tm] licenses are not reported by the *rlm_products()* call.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_detached_demox() - Remove RLM Detached Demo™ license

```
#include "license.h"
RLM_HANDLE rh;
int stat;
const char product[RLM_MAX_PRODUCT+1];
const char version[RLM_MAX_VER+1];


rh = rlm_init(...);
stat = rlm_detached_demox(rh, product, version);
```

rlm_detached_demox() requests RLM to remove an installed Detached Demo™ license. The license is specified by the product name and version.

product - the name of the product license to be removed.
version – the version of product to be removed.

Since a Detached Demo™ license can only be installed once on a particular system for any given combination of *product* and *version, rlm_detached_demox()* gives you a way to test this functionality during development.

Note: Reprise Software STRONGLY recommends that you use this function only during development, and that you do not ship products that include rlm_detached_demox() calls to your customer.

Note that rlm_detached_demox() will only remove a Detached Demo™ license created by the same version of RLM.

rlm_detached_demox() first appeared in RLM v9.3.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_diagnostics() - Print client-side diagnostics

```
RLM HANDLE rh;
char *filename;

(char *) rlm_diagnostics(rh, filename);
```

rlm_diagnostics() will print client-side diagnostics to the filename specified. *rlm_diagnostics()* can be called any time after a call to *rlm_init()* or *rlm_init_disconn()*. The values for the 3 *rlm_init()* parameters will be the values used in the most recent call to *rlm_init()*.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_errstring() - Format RLM status into a string

```
#include "license.h"
RLM_HANDLE handle;
RLM_LICENSE lic;
char string[RLM_ERRSTRING_MAX];
int stat;

(char *) rlm_errstring(lic, handle, string);
```

rlm_errstring() will take the latest status returns from any call in *handle* and from the latest rlm_checkout() call in *lic*, and format the resulting status into *string*. It is the caller's responsibility to manage the memory used by *string*. *string* should be at least RLM_ERRSTRING_MAX bytes in length. You can pass either a NULL *lic* or a NULL *handle* to *rlm_errstring()*, and only the status from the other will be returned.

Note: prior to RLM v9.1, rlm_act_errstring() was used to return a printable string corresponding to the error returned by *rlm_activate)*. Beginning in RLM v9.1, rlm_errstring() prints all RLM errors, including activation errors. Thus, *rlm_act_errstring()* is no longer required and should not be used.

rlm_errstring() returns its 3rd argument, so that it can be placed directly in an output (e.g. printf()) call.

If RLM_EXTENDED_ERROR_MESSAGES is set in the user's environment, rlm_errstring() will output additional information (for certain errors) with suggestions for solving the problem.

The returned string consists of multiple lines of information, in the following format. If any of these errors are not present, the corresponding line will not appear in the output (e.g., if there is no RLM_HANDLE error, the 2nd line will not appear):

license (RLM_LICENSE) error string (error number)
handle (RLM_HANDLE) error string (error number)
communications error (comm: error number)
system error string (errno: error number)
Optional extended error messages

For example, if a connection attempt is made to an ISV server that is not running, the following error string might be returned. Note that this example does not contain an RLM_LICENSE error line:

Networking error (in msg_init()) (-103)
Cannot connect to server (comm: -4)
Transport endpoint is not connected (errno: 146)

If RLM_EXTENDED_ERROR_MESSAGES is set, the following lines would be added to this message:

This error usually means that:
(1) The license server (rlm) is not running, or
(2) The hostname or port # in a port@host or license file is incorrect, or
(3) The ISV server isn't running, or
(4) The license server machine is down.

Note that for certain activation errors (*rlm_activate()*) additional status will be contained in the returned license string. See *rlm_activate()* for more information.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_errstring_num() - Translate RLM status value into a string

```
int error;
char string[RLM_ERRSTRING_MAX];

(char *) rlm_errstring_num(error, string);
```

rlm_errstring_num() will take any RLM status return and turn it into an error string. The *error* parameter can be the return from any RLM call which returns status (primarily rlm_stat() and rlm_license_stat())

It is the caller's responsibility to manage the memory used by *string*. *string* should be at least RLM_ERRSTRING_MAX bytes in length.

rlm_errstring() returns its 3rd argument, so that it can be placed directly in an output (e.g. printf()) call.

RLM_EXTENDED_ERROR_MESSAGES has no effect on the *rlm_errstring_num()* call.

The returned string consists of a single line of error information.

Example:

```
char string[RLM_ERRSTRING_MAX];
        rlm_errstring_max(-24 /* RLM_EL_TIMEDOUT  */, string)
        printf("RLM Error is: %s\n", string);
```

The output will be:

```
RLM Error is: License timed out by server
```

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_get_attr_health() - Check license status by testing server connection

```
#include "license.h"
RLM_LICENSE license;
int status;

status = rlm_get_attr_health(license);
```

Once you have checked out a license, you need to periodically check the health of the connection to the license server by calling *rlm_get_attr_health()* on a license-by-license basis. You can make this call as often as you like; RLM ensures that communications with the license server is done at most once every 30 seconds. This communication is called a *heartbeat*.

In general, it is sufficient to call *rlm_get_attr_health()* every couple of minutes. If you product runs for less than a few minutes, you can skip this call entirely. The main function of *rlm_get_attr_health()* is to protect against a malicious user killing and restarting the license server in order to make all licenses available again. If this is not a concern, you can simply never call *rlm_get_attr_health()* in your application.

Status of 0 indicates that everything is OK, non-zero status returns are defined in license.h

If, after successfully checking out a license, *rlm_get_attr_health()* returns a non-zero status, you should call *rlm_checkin()* on the license to free any associated memory, and then check out the license again.

If you receive a return of RLM_EL_INQUEUE from your checkout call, you would call *rlm_get_attr_health()* until you receive a 0 status, at which point the license is checked out. In this case, if *rlm_get_attr_health()* returns anything other than 0 or RLM_EL_INQUEUE, this is an error and you should call *rlm_checkin()* on that license.

If you would like RLM to provide this checking automatically (in a separate thread), see the Advanced API Options section for a description of the rlm_auto_hb() function. Note that you should call *either* *rlm_get_attr_health()* or *rlm_auto_hb()*, but not both.

Some notes on heartbeats and server status checking

Prior to RLM v10.1, when *rlm_get_attr_health()* detected an error, subsequent calls to *rlm_get_attr_health()* would return the same error without re-checking the actual status. Starting in v10.1, *rlm_get_attr_health()* will re-attempt to verify the connection to the server each time it is called. This means a few things:

- the client will be able to “re-acquire” a license that is lost due to a temporary network interruption. During the time of the interruption, *rlm_get_attr_health()* will return RLM_EL_NO_HEARTBEAT. If you are using *rlm_auto_hb(),* this is attempted 5 times, then the connection is deemed bad and it is shut down. If you are doing manual heartbeats, you control how many times you look for a heartbeat before giving up (although Reprise Software recommends that you keep this number relatively low, say 4-6 attempts).
- In *rlm_auto_hb()*, your application will not attempt to re-acquire a lost license until it has tried to verify a heartbeat 5 times. Previously, it attempted a reconnection on the initial detection of the lost heartbeat.

- In any case, if the network was interrupted and then restored, it may take more calls to *rlm_get_attr_health()* to detect a loss of heartbeat in a subsequent interruption. This is because several heartbeat responses may have been queued up for the application to read.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_get_attr_lfpath() - Get license path in use by RLM

```
#include "license.h"
RLM_LICENSE license;
char *path;

path = rlm_get_attr_lfpath(license);
```

Once you have attempted a license checkout, you can determine the license path in use by RLM by calling *rlm_get_attr_lfpath()* on the license handle (note: the checkout does not need to be successful for *rlm_get_attr_lfpath()* to work). This call will retrieve the same path for any license handle passed in.

You should ***NOT*** free the returned string.

Note: on Windows, the path components are separated by the ';' character. On all other RLM platforms, the path components are separated by the ':' character.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_get_rehost() - Retrieve the hostid of a rehostable license.

```
#include "license.h"
RLM_HANDLE handle;
char *product;
char *hostid[RLM_MAX_HOSTID_STRING+1];
int status;

status = rlm_get_rehost(handle, product, hostid);
```

rlm_get_rehost() will return the hostid for the specified *product* if there is a rehostable hostid on this system. If status==0, *hostid* will contain the hostid string for this *product*.

This call can be used to retrieve a rehostable hostid when the license file is lost, and then transmit this hostid to the activation server to retrieve the hostid.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_hostid(), rlm_all_hostids(), rlm_all_hostids_free() - retrieve the hostid of this machine.

rlm_hostid()

```
#include "license.h"
RLM_HANDLE handle;
int type;
char hostid[RLM_MAX_HOSTID_STRING];
const char *description;

description = rlm_hostid(handle, type, hostid);
```

Call rlm_hostid() on any open RLM_HANDLE to retrieve the hostid of type **type**. The hostid will be returned in the string **hostid**.

The value of **type** should be one of:

RLM_HOSTID_32BIT
RLM_HOSTID_DISKSN (Windows only)
RLM_HOSTID_ETHER
RLM_HOSTID_USER
RLM_HOSTID_HOST
RLM_HOSTID_IP
RLM_HOSTID_RLMID1

or one of your ISV-defined hostid types.

Note: **type** could also be one of RLM_HOSTID_ANY, RLM_HOSTID_DEMO, or RLM_HOSTID_STRING, but these will always return "ANY", "DEMO", or "".

The description return value will be NULL for an error, otherwise it is a static string - do not free it. Currently it is always an empty string, but may be used in the future.

Note: You cannot retrieve a rehostable hostid with rlm_hostid() or rlm_all_hostids(). Call rlm_get_rehost() to retrieve a rehostable hostid for a product.

rlm_all_hostids()

The rlm_all_hostids() call returns a list of hostids for hostid types which allow for multiple instances on a given machine.

```
RLM_HANDLE handle;
int type;
char **list;
```

list = ***rlm_all_hostids*(handle, type)**;

rlm_all_hostids() returns a pointer to an array of (char *) pointers. Each pointer points to a string which is one instance of the specified hostid type. The list is terminated with a NULL pointer.

rlm_all_hostids_free()

Free all memory allocated for the list with the *rlm_all_hostids_free()* call.

char **list;
(void) ***rlm_all_hostids_free(list)**;*

Example:

```
char **list, **list_save;

    list_save = list = rlm_all_hostids(handle, RLM_HOSTID_ETHER);
    while (list && *list)
    {
         printf("Hostid: %s\n", *list);
         list++;
    }
    rlm_all_hostids_free(list_save);
```

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_init() - Initialize licensing operations with RLM

```
#include "license.h"
RLM_HANDLE handle;
const char *license;
const char *argv0;
const char *license_strings;
int promise;

handle = rlm_init(license, argv0, license_strings);
```

Before any RLM operations can be done, you must call either *rlm_init* or *rlm_init_disconn* to obtain an RLM_HANDLE. This handle is then passed to the *rlm_checkout()*, *rlm_stat()*, *rlm_errstring()*, and *rlm_close()* calls.

The first parameter is the license file (or directory) you would like to use. If you allow the license administrator to specify the license file, put the path to this license file here. If you do not allow specification of the license file, Reprise Software recommends searching the current directory - you do this by passing a string with a single dot (".") as the first parameter. This first parameter can be a list, starting in RLM v11.1, however it must be a single file or directory prior to v11.1 This parameter can also be a **port@host** specification. Beginning in RLM v11.0, this string must have a length <= RLM_MAX_PATH bytes (1024 on Unix, 2048 on windows), otherwise an RLM_EH_BADPARAM error will be returned by *rlm_init()*.

The second parameter should be your argv[0] invocation argument. This will cause RLM to look in the directory where your binary resides to find license files. If you do not have access to argv[0], pass a NULL or empty string as the second parameter. Using anything other than an empty/NULL string or argv[0] will result in unpredictable behavior. Beginning in RLM v11.0, this string must have a length <= RLM_MAX_PATH bytes (1024 on Unix, 2048 on windows), otherwise an RLM_EH_BADPARAM error will be returned by *rlm_init()*.

The third parameter is used to pass licenses into RLM directly. This can be one license, or a list of licenses separated by the path separator (':' on Unix, ';' on Windows). Each license must be enclosed within angle brackets ('<' and '>'). This would be used, for example, when you are licensing a library and you want to give your customer a license to use the library yet you do not want to require that they use a separate license file. In this case, they would compile the license into the code and you would pass it into rlm_init() in this parameter. ***Do not include HOST or ISV lines in this license, only the LICENSE line. Note that these licenses must be node-locked, uncounted (or SINGLE) licenses.***

> ***NOTE: On Windows platforms except for x86_w1, if the paths your application would pass to rlm_init() in the first and second parameters are Unicode wide characters (wchar_t or WCHAR), you must first convert them to UTF-8. The Win32 function WideCharToMultiByte() can be used for this conversion.***

For example, to pass 2 licenses into rlm using the *rlm_init()* call, pass a string similar to following as the third parameter to rlm_init() (note – you must include the entire signed license within the angle brackets):

```
<LICENSE isv lic1 1.0 permanent 0 key hostid=xxx .sig=yyy ..>:<LICENSE isvname lic2 1.0 permanent 0 key
hostid=xxx  sig=yyy ...>
```

Note that RLM uses environment variables for a number of user-selectable options, such as queuing (RLM_QUEUE), license roaming (RLM_ROAM), project identification (RLM_PROJECT), etc. It is

possible for you as an ISV to set these environment variables within your application, but if you wish to do this, you should do it before you call *rlm_init()*, because the environment is read and initialized at the time *rlm_init()* is called.

Retrieve the status of the *rlm_init()* call by calling *rlm_stat(handle)* and providing the *handle* returned by *rlm_init()*: For a list of status returns, see *Appendix A - RLM Status Returns* on page **137**.

```
int status;
status = rlm_stat(handle);
```

rlm_init() will set up a list of licenses, port@host specifications and license files in the RLM handle. This order will determine the order in which license checkouts will be attempted. The order will be randomized if the user has set the RLM_PATH_RANDOMIZE environment variable to any value. The default order is:

- the contents of the *ISV*_LICENSE (if present) or RLM_LICENSE environment (note that if using *ISV*_LICENSE, the *ISV* part of the name must be in the same case as was entered in rlm_isv_config.c – generally lower case. "LICENSE" must be uppercase).
- the license specifications in the first parameter (*license*) in the *rlm_init()* call.
- the license files contained in the directory (*argv0*) in the second parameter in the *rlm_init()* call
- any licenses passed as strings in the third parameter (*license_strings*) in the *rlm_init()* call.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_license_XXXX() - Get checked-out license information.

This is a family of functions that operate on a valid license handle. These functions return status and attributes of a checked-out license. Note that in the case of token-based licenses, these data will be attributes of the license which actually satisfied the request, rather than attributes of the token-based license itself. These functions are divided into *policy* functions, which you should use to affect license policy, and *display* functions, which you should use only to display license data to your user.

> WARNING: Other than the *policy* functions, be very careful using these functions. The *display* functions are intended for the application to use to determine the details of the license checked out, for the purposes of display to the user. Use of these functions to affect the behavior of the application based on the contents of the optional fields in the license is annoying and frustrating to license administrators. This is because doing so makes application behavior mysterious to them, and non-standard across all their licensed applications. For example, using rlm_license_customer() to display the name of the customer is reasonable. Making runtime decisions about application behavior or capability based on the data returned from rlm_license_customer() makes the application behavior different from other licensed applications and risks customer dissatisfaction. This violates the RLM design philosophy of "policy in the license", and is historically a sore point with license administrators of license management systems. Reprise Software urges you to take heed.

All functions operate on an RLM_LICENSE. Definitions for all functions are:

```
#include "license.h"
RLM_LICENSE license;
```

Note: On all of the following functions that return strings, if there is no valid checked-out license or the license handle is invalid, the function returns a NULL pointer. For functions that return int, a return value of RLM_EL_NOHANDLE indicates that a null or invalid handle was passed to the function.

The functions are:

Policy Functions

- **rlm_license_akey() - Retrieve activation key used to create license**

  char *akey = *rlm_license_akey(license);*
  Note: The akey field is only used by RLM to control license pooling in the server.

- **rlm_license_count() - Retrieve requested license count**

  int count = *rlm_license_count(license);*
  Note: The count is the count you requested in checkout.

- **rlm_license_stat() - Retrieve license status**

int status;
status = *rlm_license_stat(license);*

You can retrieve the status of an *rlm_checkout()* call by calling *rlm_license_stat(license)*. *license* is the license handle returned by *rlm_checkout()*. This call does not query the license server for the status, it merely returns the status stored the last time the server was contacted. You can call this as often as you like. For a list of status returns, see Appendix B – RLM Status Values on page 137. This call, and *rlm_license_goodonce()* are the only calls in the family of *rlm_license_xxxx()* functions which you should use to affect application behavior.

Note: you cannot call *rlm_license_stat()* on a license handle after that handle has been checked in, or if the RLM_HANDLE used to check it out has been closed. This will result in unpredictable behavior (including possible application crashes), since the handle you are using has been freed by RLM. :q

- **rlm_license_goodonce() - Was checkout ever successful on this handle**

  int status;
  status = *rlm_license_goodonce(license);*

  You can determine whether a checkout was ever successful on a particular license handle by calling *rlm_license_goodonce(license)*. If *status* is 0, the checkout was never successful. If non-zero, the checkout succeeded at one time (although the license may no longer be valid). Note that a license status of RLM_EL_OVERSOFT will be considered to be a good checkout, but RLM_EL_INQUEUE is not. RLM_EL_OVERSOFT is an error if you have a misconfigured token-based license (see the note in the token-based license restrictions section).

- **rlm_license_options() - Retrieve license options**

  char *options = *rlm_license_options(license);*

  The meaning of the options string is completely determined by an individual ISV. This string is intended to encode product options for this license.

Display Functions

- **rlm_license_contract() - Retrieve license contract string**

  char *contract = *rlm_license_contract(license);*
  Note: This license field is unused by RLM.

- **rlm_license_customer() - Retrieve license customer string**

char *customer = *rlm_license_customer(license);*
Note: This license field is unused by RLM.

- **rlm_license_detached_demo() - Retrieve ”detached demo” status of license.**

  int detached = *rlm_license_detached_demo(license);*
  If this is a Detached Demo™ license, the return is 1, otherwise 0.

- **rlm_license_exp() - Retrieve license expiration date**

  char *exp = *rlm_license_exp(license);*
  Note: For licenses checked-out from a license server, the expiration date returned by the server is the first (earliest) expiration date from all the licenses which make up the license pool used to satisfy this request. In other words, there may be other licenses for this same product which expire later than this date.

- **rlm_license_exp_days() - Retrieve the # of days until license expiration**

  int days = *rlm_license_exp_days(license);*

  Note: For licenses checked-out from a license server, the number of days to expiration is based on the first (earliest) expiration date from all the licenses which make up the license pool used to satisfy this request. In other words, there may be other licenses for this same product which expire later than this date.

  Also Note: *rlm_license_exp_days()* counts today as a day. So, for example, a license which expires tomorrow at midnight will be reported as expiring in 2 days. A license which expires today at midnight will be reported as expiring in 1 day.

  If days == 0, this is a permanent license. If days is < 0, there was an error.

- **rlm_license_hostid() - Retrieve license hostid**

  char *hostid = *rlm_license_hostid(license);*.

- **rlm_license_issued() - Retrieve license issued date**

  char *issued = *rlm_license_issued(license);*

  (Note: this value is only correct for licenses which aren't served. Any license coming from a license server has an undefined rlm_license_issued() value.)

- **rlm_license_issuer() - Retrieve license issuer string**

char *issuer = *rlm_license_issuer(license);*
Note: This license field is unused by RLM.

- **rlm_license_line_item() - Retrieve license line_item string**

  char *line_item = *rlm_license_line_item(license);*
  Note: This license field is unused by RLM.

- **rlm_license_platforms() - Retrieve licensed platforms**
  char *platforms= *rlm_license_platforms(license);*

- **rlm_license_product() - Retrieve licensed product**

  char *product = *rlm_license_product(license);*
  Note that rlm_license_product() retrieves the product name which satisfied the request. This may be different than the product requested. In the case of token-based licenses, the license requested is not the product that satisfies the license request. The actual product which satisfied the request is returned by rlm_license_product(). Also note that only the attributes of the **first** license (in the case of a compound token-based license) is returned by these calls. The first license is the first license listed in the token definition.

- **rlm_license_single() - Is license a "single" type**
  int single = *rlm_license_single(license);*

  Returns 1 if the license is "single", 0 otherwise.

- **rlm_license_start() - Retrieve license start date**

  char *start= *rlm_license_start(license);*

- **rlm_license_type() - Retrieve license type**

  int type = *rlm_license_type(license);*
  The type variable has bits set for the specified license types, as defined in *license.h*:

  - RLM_LA_BETA_TYPE - "beta" specified in license type keyword
  - RLM_LA_EVAL_TYPE - "eval" specified in license type keyword
  - RLM_LA_DEMO_TYPE - "demo" specified in license type keyword
  - RLM_LA_SUBSCRIPTION_TYPE - "subscription" specified in license type keyword

  Note: This license field is unused by RLM.

- **rlm_license_tz() - Retrieve license timezone spec**

  int tz = *rlm_license_tz(license);*

- **rlm_license_uncounted() - Is license uncounted**

  int uncounted = rlm_license_uncounted(license);

  Returns 1 if the license is uncounted, 0 otherwise.

- **rlm_license_ver() - Retrieve license version**

  char *ver= *rlm_license_ver(license);*
  Note that rlm_license_ver() returns the actual version of the license that was used to satisfy the request. This may be different than the version requested.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_products() - Generate list of products that can be checked out

```
#include "license.h"
RLM_PRODUCTS products;
RLM_HANDLE handle;
char *product;
char *ver;
int status;

products = rlm_products(handle, product, ver);
(void) rlm_product_first(products);
status = rlm_product_next(products);
```

NOTE: rlm_products() is an expensive call. If you don't absolutely need it, don't call it. If you do call it, specify a product name if you can. You should avoid calling it more than once inside an application. Why is it expensive? If called with empty product/version strings, it has to validate the license keys for all node-locked uncounted/single-use licenses in local license files.

rlm_products generates a list of products of the specified version that can be checked out. If *product* is an empty or NULL string, all products are checked. If *ver* is empty or NULL, any version will be listed. If *rlm_products()* returns a non-null pointer, then there are products in the list. The **status** return from *rlm_product_next()* is 0 if there is another product in the list, or -1 if the list is exhausted. *rlm_products()* does not report on D*etached Demo*[tm] licenses.

To examine the list of products, first call *rlm_products()* to retrieve the products pointer. Next, use *rlm_product_first()* and *rlm_product_next()* to walk the list of products returned. At any point after calling rlm_product_first(), you can call the appropriate function below. Note: you should **not** free any data returned by any of these calls.

Prior to RLM v4.0BL2, *rlm_products()* returned the license information in the same order that the license files were present in the license file path. However, starting in RLM v4.0BL2, *rlm_products()* returns the licenses in the same order that rlm_checkout() will attempt checkouts. This order is:

- If **RLM_ROAM** is set to a positive value, roamed licenses on the local node first,
- All node-locked, uncounted licenses in local license files (from all license files in the license file path) will be next
- All licenses served by servers are next,
- Finally, if **RLM_ROAM** is not set, the local roamed licenses will be last.

Note that *rlm_products()* returns the list of valid roamed products on the local node, ***whether or not it can check out an rlm_roam license.***

Note that rlm_checkout() first processes licenses from connected servers, then it attempts checkouts from servers that are not connected. However, rlm_products() will connect to all servers and get the lists from each of them. It will then close connections to all servers that have no active licenses checked out. If your software depends on the order of the licenses on license servers as returned from rlm_products() [NOTE: Reprise Software does not recommend this], then you **should** call rlm_set_attr_keep_conn(handle, 1) before calling rlm_products(), so that rlm_products() will not close any connections that it makes.

char ***rlm_product_name(products)** - returns the product name.

char ***rlm_product_ver(products)*** - returns the product version.
char ***rlm_product_exp(products)*** - returns the expiration date. If this product represents a pool in a license server, the expiration date will be the *earliest* expiration of any of the licenses which were combined to create the pool.

int ***rlm_product_exp_days(products)*** - returns the number of days until expiration. Note that "0" indicates a permanent license; a license which expires today has a value of 1. If this product represents a pool in a license server, the expiration date will be the *earliest* expiration of any of the licenses which were combined to create the pool. ***rlm_product_exp_days()*** is new in RLM v10.0.

char ***rlm_product_akey(products)*** - returns the akey= attribute. New in v11.0.
char ***rlm_product_customer(products)*** - returns the customer attribute. New in v10.0.
char ***rlm_product_contract(products)*** - returns the contract attribute. New in v10.0.
int ***rlm_product_count(products)*** - returns the license count.

char ***rlm_product_hostid(products)*** - returns the license nodelock hostid, if it exists. New in v12.0.
int ***rlm_product_isalias(products)*** - returns non-zero for an alias license, 0 otherwise. New in v14.1
int ***rlm_product_isnodelocked(products)*** - returns non-zero for a nodelocked license. New in v14.1
int ***rlm_product_issingle(products)*** - returns non-zero for a *single* license, 0 otherwise. New in v14.1
char ***rlm_product_issuer(products)*** - returns the issuer attribute. New in v10.0.
char ***rlm_product_options(products)*** - returns the product options.
int ***rlm_product_tz(products)*** - returns the license timezone specification.
int ***rlm_product_type(products)*** - returns the license type (TYPE= parameter).
(Note: the license type flags (RLM_LA__xxx_TYPE) are defined in license.h)

Note: the list of products may contain products that cannot be checked out at any given time, in the case of a SINGLE license that is in use.. It is possible (at some time) to check out every product in the list, however. In other words, the list contains only licenses for which the license key is good, the time is past the start date and before the expiration date, the timezone is correct, and we are on the correct host.

Also note that the following licenses will *never* be returned by rlm_products():

- D*etached Demo*^{tm} licenses
- licenses passed in the 3rd parameter to rlm_init()

The data returned by the *rlm_products()* call is dynamically allocated. Call *rlm_products_free(products)* to free this memory when you are finished with it, in order to avoid memory leaks in your program. You should only call *rlm_products_free()* once on the data returned by *rlm_products()*, and only when you are finished accessing this data.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_putenv() - Set environment variable within the application

```
#include "license.h"
RLM_HANDLE rh;
int status;
const char *nvp;

status = rlm_putenv(const char *nvp);
```

rlm_putenv() sets the specified name in to the specified value in the process's environment. The return of *rlm_putenv()* is the return of the system *putenv()* call.

Example:

```
const char *nvp = “RLM_ROAM=10”;
rlm_putenv(nvp);
```

In this example, the environment value of RLM_ROAM is set to 10.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_set_environ() - Set user/host/ISV-defined values for RLM

```
#include "license.h"
RLM_HANDLE rh;
char user[RLM_MAX_USERNAME+1];
char host[RLM_MAX_HOSTNAME+1];
char isv[RLM_MAX_ISVDEF+1];

rlm_set_environ(handle, user, host, isv);
```

License sharing operates by comparing user, host, and ISV fields for matches. *rlm_set_environ()* allows the ISV to override the system's notion of user and/or host, and also provides a way to set the ISV-defined data. If any of user/host/isv are passed in as NULL, the corresponding field remains unchanged.

The ISV field should be a printable string which does not contain the double-quote character (").

Note that *rlm_set_environ()* should be called after *rlm_init()* and before any *rlm_checkout()* call to which it should apply. Once *rlm_checkout()* is called, these values will persist for the life of the RLM_HANDLE in which you call *rlm_set_environ()*.

Starting in RLM v9.1, you can call *rlm_set_environ()* after the first *rlm_checkout()* call, and the new user, host, and ISV-defined parameters will apply to the new checkout. The original settings will continue to apply to *rlm_products()* calls, however.

Note that RLM always treats usernames and hostnames as case-insensitive.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_set_attr_reference_hostid() - Set reference hostid for actpro

```
#include "license.h"
RLM_HANDLE rh;
char *reference_hostid;

(void) rlm_set_attr_reference_hostid(handle, reference_hostid);
```

Beginning in RLM v12.3, you can set the hostid which RLM uses as a reference hostid when creating a rehostable hostid. It is important that the hostid you set is a valid RLM hostid which is valid on the current host, otherwise your rehostable hostid will not work and will always return RLM_EL_NOTTHISHOST.

You can call *rlm_set_attr_reference_hostid()* any time before an *rlm_act_request(), rlm_activate()* or *rlm_act_revoke_reference()* call.

If you set the reference hostid when creating a rehostable hostid, you must set the same hostid before calling *rlm_act_revoke_reference()*, otherwise the rehostable hostid will not be revoked.

The hostid string you pass to this function must be <= RLM_MAX_HOSTID_STRING characters long***, and must be a valid hostid on the current host.***

Note that RLM selects a reference hostid automatically, and you should never need to make this call.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_set_attr_req_opt() - Set required substring in license options

```
#include "license.h"
RLM_HANDLE rh;
char *opts;

rlm_set_attr_req_opt(handle, opts);
```

Beginning in RLM v12.0, you can request that any license must contain a certain substring in the "options=" field.

You can call *rlm_set_attr_req_opt()* any time before an *rlm_checkout()* or *rlm_products()* call, and the value of the option substring can be changed for subsequent requests. Note that if you set **opts** to an empty string (""), no checking of the license options will be done by *rlm_checkout()* or *rlm_products()*.

The opts parameter must be a substring in the license options, and the comparison is ***CASE SENSITIVE.***

Note: once you call *rlm_set_attr_req_opt()*, you will only see licenses with the specified substring in the options field in either the *rlm_checkout()* or *rlm_products()* calls. If you want to see other licenses, call *rlm_set_attr_req_opt()* with an empty string.

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_sign_license() - Sign an individual license in-memory

```
#include "license.h"
RLM_HANDLE rh;
int encode_bits;
char *hostid;
char license[RLM_MAX_LINE+1];

status = rlm_sign_license(rh, encode_bits, hostid, license);
```

rlm_sign_license runs the internal signature algorithm to compute the license key for the license string found in **license**.

rlm_sign_license() should be called with a valid RLM_HANDLE as its first parameter.

The 2nd parameter - *encode_bits* - indicates the key encoding desired. Valid values are:

- 4 - encode license key 4 bits/character - this produces HEX numbers
- 5 - encode license key 5 bits/character - this produces all UPPERCASE license keys
- 6 - encode license key 6 bits/character - this produces license keys in mixed-case

If you specify a value that is < 4 or > 6, 4 bits/character will be used.

The 3rd parameter - *hostid* - is the hostid of the license server, if this is a floating license. You should pass an empty or NULL string if this is a node-locked license.

The 4th parameter - *license* - should contain a valid RLM license, with the signature replaced with the string "sig". On successful completion, the "sig" string will be replaced with the correct license signature in this string. Note that this string should contain only the (single) LICENSE line, not the HOST and ISV lines.

A successful call to *rlm_sign_license()* will return a 0 status. Any other status return indicates an error, and the license will not be valid.

Example – sign a nodelocked license:

```
#include "license.h"
RLM_HANDLE rh;
char license[RLM_MAX_LINE+1];

    rh = rlm_init((char *)NULL, (char *)NULL, (char *)NULL);
    if (!rh)
        -error-
    else
    {
      (void) strcpy(license,
        "LICENSE demo rlmclient 1.0 12-apr-2019 uncounted  hostid=ANY options=xyz sig");

      status = rlm_sign_license(rh, 6, (char *) NULL, license);
    }
```

Back to Appendix A – RLM Embedded API.

Table of Contents

rlm_stat() - Retrieve RLM_HANDLE status

```
#include "license.h"
RLM_HANDLE handle;
int status;

status = rlm_stat(handle);
```

rlm_stat() retrieves the status of the handle created with the rlm_init() call. For a list of status returns, see Appendix B – RLM Status Values on page 137.

Back to Appendix A – RLM Embedded API.

Table of Contents

Appendix B – RLM Status Values

The API functions return status (via the rlm_stat() and *rlm_license_stat()* (see rlm_license_XXXX()) calls

Note that this is a complete list of all RLM errors. RLM Embedded programs will not generate many of these errors, since many errors listed here relate to the license server.

***rlm_stat()* returns general RLM_HANDLE errors. These are:**

0	0	Success
RLM_EH_NOHANDLE	-101	No handle supplied to call
RLM_EH_READ_NOLICENSE	-102	Can't read license data
RLM_EH_NET_INIT	-103	Network (msg_init()) error
RLM_EH_NET_WERR	-104	Error writing to network
RLM_EH_NET_RERR	-105	Error reading from network
RLM_EH_NET_BADRESP	-106	Unexpected response
RLM_EH_BADHELLO	-107	HELLO message for wrong server
RLM_EH_BADPRIVKEY	-108	Error in private key
RLM_EH_SIGERROR	-109	Error signing authorization
RLM_EH_INTERNAL	-110	Internal error
RLM_EH_CONN_REFUSED	-111	Connection refused at server (this can also happen if you have a bad TCP/IP address in your local database)
RLM_EH_NOSERVER	-112	No server to connect to
RLM_EH_BADHANDSHAKE	-113	Bad communications handshake
RLM_EH_CANTGETETHER	-114	Can't get ethernet address
RLM_EH_MALLOC	-115	malloc() error
RLM_EH_BIND	-116	bind() error
RLM_EH_SOCKET	-117	socket() error
RLM_EH_BADPUBKEY	-118	Error in public key
RLM_EH_AUTHFAIL	-119	Authentication failed
RLM_EH_WRITE_LF	-120	Can't write new license file
RLM_EH_DUP_ISV_HID	-122	ISV-defined hostid already registered
RLM_EH_BADPARAM	-123	Bad parameter passed to RLM function
RLM_EH_ROAMWRITEERR	-124	Roam File write error
RLM_EH_ROAMREADERR	-125	Roam File read error
RLM_EH_HANDLER_INSTALLED	-126	Heartbeat handler already installed
RLM_EH_CANTCREATELOCK	-127	Can't create 'single' lockfile
RLM_EH_CANTOPENLOCK	-128	Can't open 'single' lockfile
RLM_EH_CANTSETLOCK	-129	Can't set lock for 'single'
RLM_EH_BADRLMLIC	-130	Bad/missing/expired RLM license
RLM_EH_BADHOST	-131	bad hostname in license file or port@host
RLM_EH_CANTCONNECTURL	-132	Can't connect to specified URL (activation)

RLM_EH_OP_NOT_ALLOWED	-133	Operation not allowed on server. The status, reread, shutdown, or remove command has been disabled for this user.
RLM_EH_ACT_BADSTAT	-134	Bad status return from Activation server
RLM_EH_ACT_BADLICKEY	-135	Activation server built with incorrect license key
RLM_EH_ACT_BAD_HTTP	-136	Error in HTTP transaction with Activation server
RLM_EH_DEMO_EXISTS	-137	Demo already created on this system
RLM_EH_DEMO_WRITEERR	-138	Demo install file write error
RLM_EH_NO_DEMO_LIC	-139	No "rlm_demo" license available
RLM_EH_NO_RLM_PLATFORM	-140	RLM is unlicensed on this platform
RLM_EH_EVAL_EXPIRED	-141	The RLM evaluation license compiled into this binary has expired
RLM_EH_SERVER_REJECT	-142	Server rejected (too old)
RLM_EH_UNLICENSED	-143	Unlicensed RLM option
RLM_EH_SEMAPHORE_FAILURE	-144	Seamphore initialization failure
RLM_EH_ACT_OLDSERVER	-145	Activation server too old (doesn't support encryption)
RLM_EH_BAD_LIC_LINE	-146	Invalid license line in LF
RLM_EH_BAD_SERVER_HOSTID	-147	Invalid hostid on SERVER line
RLM_EH_NO_REHOST_TOP_DIR	-148	No rehostable hostid top-level dir
RLM_EH_CANT_GET_REHOST	-149	Cannot get rehostable hostid
RLM_EH_CANT_DEL_REHOST	-150	Cannot delete rehostable hostid
RLM_EH_CANT_CREATE_REHOST	-151	Cannot create rehostable hostid
RLM_EH_REHOST_TOP_DIR_EXISTS	-152	Rehostable top directory exists
RLM_EH_REHOST_EXISTS	-153	Rehostable hostid exists
RLM_EH_NO_FULFILLMENTS	-154	No fulfillments to revoke
RLM_EH_METER_READERR	-155	Meter read error
RLM_EH_METER_WRITEERR	-156	Meter write error
RLM_EH_METER_BADINCREMENT	-157	Bad meter increment command
RLM_EH_METER_NO_COUNTER	-158	Can't find counter in meter
RLM_EH_ACT_UNLICENSED	-159	Activation Unlicensed
RLM_EH_ACTPRO_UNLICENSED	-160	Activation Pro Unlicensed
RLM_EH_SERVER_REQUIRED	-161	Counted license requires server
RLM_EH_DATE_REQUIRED	-162	REPLACE license requires date
RLM_EH_NO_METER_UPGRADE	-163	METERED licenses can't be UPGRADED
RLM_EH_NO_CLIENT	-164	Disconnected client data can't be found
RLM_EH_NO_DISCONN	-165	Operation not allowed on disconnected handle
RLM_EH_NO_FILES	-166	Too many open files
RLM_EH_NO_BROADCAST_RESP	-167	No response to broadcast message
RLM_EH_NO_BROADCAST_HOST	-168	Broadcast response didn't include hostname
RLM_EH_SERVER_TOO_OLD	-169	Server too old for disconnected operations
RLM_EH_BADLIC_FROM_SERVER	-170	License from server doesn't authenticate
RLM_EH_NO_LIC_FROM_SERVER	-171	No License returned from server
RLM_EH_CACHEWRITEERR	-172	Client Cache File write error

RLM_EH_CACHEREADERR	-173	Client Cache File read error
RLM_EH_LIC_WITH_NEW_KEYWORDS	-174	License returned from server has keywords I don't understand
RLM_EH_NO_ISV	-175	Unknown ISV name
RLM_EH_NO_CUSTOMER	-176	Unknown Customer name
RLM_EH_NO_SQL	-177	Cannot open MySQL database (RLMCloud only)
RLM_EH_ONLY_LOCAL_CMDS	-178	Only local command-line commands allowed
RLM_EH_SERVER_TIMEOUT	-179	Server timeout on read
RLM_EH_NONE_SIGNED	-180	rlmsign did not sign any licenses (warning)
RLM_EH_DUP_XFER	-181	Duplicate disconnected transfer
RLM_EH_BADLOGIN	-182	Bad/No login credentials to server
RLM_EH_WS_NOSUPP	-183	Function not supported with web services
RLM_EH_NOFUNC	-184	Function not available
RLM_EH_TOOMUCHJSON	-185	JSON reply too big
RLM_EH_NOLICFROMSERV	-186	Server returned no temp license
RLM_EH_TEMPFROMCLOUD	-187	Temporary licenses come from RLMCloud servers only
RLM_EH_NOTTEMP	-188	License is not a temporary license
RLM_EH_NOLICENSE	-189	No license supplied to call
RLM_EH_NOTEMPFROMLOCAL	-190	Local license can't be converted to temp license
RLM_EH_NOHTTPSSUPPORT	-191	Actpro HTTPS support not available
RLM_EH_NOHTTPSDATA	-192	No returned data from HTTPS
RLM_EH_NOTTHISHOST	-193	Wrong Host
RLM_EH_NOTRANSBIN	-194	Translated binaries not supported (mac)

In addition, rlm_activate() will return the following errors:I

RLM_ACT_BADPARAM	-1001	Unused – RLM_EH_BADPARAM returned instead.
RLM_ACT_NO_KEY	-1002	No activation key supplied
RLM_ACT_NO_PROD	-1003	No product definition exists
RLM_ACT_CANT_WRITE_KEYS	-1004	Can't write keyf table
RLM_ACT_KEY_USED	-1005	Activation key already used
RLM_ACT_BAD_HOSTID	-1006	Missing hostid
RLM_ACT_BAD_HOSTID_TYPE	-1007	Invalid hostid type
RLM_ACT_BAD_HTTP	-1008	Bad HTTP transaction. Note: unused after v3.0BL4
RLM_ACT_CANTLOCK	-1009	Can't lock activation database
RLM_ACT_CANTREAD_DB	-1010	Can't read activation database
RLM_ACT_CANT_WRITE_FUFILL	-1011	Can't write licf table
RLM_ACT_CLIENT_TIME_BAD	-1012	Clock bad on client system (not within 7 days of server)
RLM_ACT_BAD_REDIRECT	-1013	Can't write licf table
RLM_ACT_TOOMANY_HOSTID_CHANGES	-1014	Too many hostid changes for refresh-type activation

RLM_ACT_BLACKLISTED	-1015	Domain on blacklist for activation
RLM_ACT_NOT_WHITELISTED	-1016	Domain not on activation key whitelist
RLM_ACT_KEY_EXPIRED	-1017	Activation key expired
RLM_ACT_NO_PERMISSION	-1018	HTTP request denied (this is a setup problem)
RLM_ACT_SERVER_ERROR	-1019	HTTP internal server error (usually a setup problem)
RLM_ACT_BAD_GENERATOR	-1020	Bad/missing generator file (Activation Pro)
RLM_ACT_NO_KEY_MATCH	-1021	No matching activation key in database
RLM_ACT_NO_AUTH_SUPPLIED	-1022	No proxy authorization supplied
RLM_ACT_PROXY_AUTH_FAILED	-1023	Proxy authentication failed
RLM_ACT_NO_BASIC_AUTH	-1024	No basic authentication supported by proxy
RLM_ACT_GEN_UNLICENSED	-1025	Activation generator unlicensed (ISV_mklic)
RL_ACT_DB_READERR	-1026	Activation database read error (Activation Pro)
RLM_ACT_GEN_PARAM_ERR	-1027	Generating license - bad parameter
RLM_ACT_UNSUPPORTED_CMD	-1028	Unsupported command to license generator
RLM_ACT_REVOKE_TOOLATE	-1029	Revoke command after expiration
RLM_ACT_KEY_DISABLED	-1030	Activation key disabled
RLM_ACT_KEY_NO_HOSTID	-1031	Key not fulfilled on this hostid
RLM_ACT_KEY_HOSTID_REVOKED	-1032	Key revoked on this hostid
RLM_ACT_NO_FULFILLMENTS	-1033	No fulfillments to remove
RLM_ACT_LICENSE_TOOBIG	-1034	Generated license too long
RLM_ACT_NO_REHOST	-1035	Counted licenses can't be rehostable
RLM_ACT_BAD_URL	-1036	License Generator not found on server
RLM_ACT_NO_LICENSES	-1037	RLMCloud: No licenses found
RLM_ACT_NO_CLEAR	-1038	Unencrypted requests not allowed
RLM_ACT_BAD_KEY	-1039	Bad activation key (illegal char)
RCC_CANT_WRITE_FULFILL	-1040	RLMCloud: Can't write licf table
RCC_PORTAL_CANT_WRITE_FULFILL	-1041	RLMCloud: Can't write licf table
RLM_ACT_KEY_TOOMANY	-1042	Insufficient count left in activation key
RLM_ACT_SUB_BADTYPE	-1043	Subscription license not Nodelocked or Single
RLM_ACT_CONTACT_BAD	-1044	Contact information supplied is bad

***rlm_license_stat()* returns RLM_LICENSE errors and status. These are:**

Status	Value	Meaning	Full Description
0	0	Success	
RLM_EL_NOPRODUCT	-1	No authorization	rlm_checkout() did not find a product to satisfy

		for product	your request.
RLM_EL_NOTME	-2	Authorization is for another ISV	The license you are requesting is in the license file, but it is for a different ISV.
RLM_EL_EXPIRED	-3	Authorization has expired	The only license available has expired. This error will only be returned for local license lines, never from a license server.
RLM_EL_NOTTHISHOST	-4	Wrong host for authorization	The hostid in the license doesn't match the hostid of the machine where the software is running.
RLM_EL_BADKEY	-5	Bad key in authorization	The signature in the license line is not valid, i.e. it does not match the remainder of the data in the license.
RLM_EL_BADVER	-6	Requested version not supported	Your application tried to check out a license at a higher version than was available, e.g., you specified v5, but the available license is for v4.
RLM_EL_BADDATE	-7	bad date format - not permanent or dd-mm-yy	The expiration, start, or issued date wasn't understood, eg, 316-mar-2010 or 31-jun-2010. You'd probably never see this in the field unless somebody had tampered with the license file.
RLM_EL_TOOMANY	-8	checkout request for too many licenses	Your checkout request will never work, because you have asked for more licenses than are issued.
RLM_EL_NOAUTH	-9	No license auth supplied to call	This is an internal error.
RLM_EL_ON_EXC_ALL	-10	On excludeall list	The license administrator has specified an EXCLUDEALL list for this product, and the user (host, etc) is on it.
RLM_EL_ON_EXC	-11	On feature exclude list	The license administrator has specified an EXCLUDE list for this product, and the user (host, etc) is on it.
RLM_EL_NOT_INC_ALL	-12	Not on the includeall list	The license administrator has specified an INCLUDEALL list for this product, and you are not on it.
RLM_EL_NOT_INC	-13	Not on the feature include list	The license administrator has specified an INCLUDE list for this product, and you are not on it.
RLM_EL_OVER_MAX	-14	Request would go over license MAX	The license administrator set a license MAX usage option for a user or group. This checkout request would put this user/group/host over that limit.
RLM_EL_REMOVED	-15	License (rlm)removed by server	A license administrator removed this license using the rlmremove command or the RLM web interface.
RLM_EL_SERVER_BADRESP	-16	Unexpected response from server	The application received a response from the license server which it did not expect. This is an internal error.
RLM_EL_COMM_ERROR	-17	Error communicating with server	This indicates a basic communication error with the license server, either in a network initilization, read, or write call.
RLM_EL_NO_SERV_SUPP	-18	License server	

		doesn't support this	
RLM_EL_NOHANDLE	-19	No license handle	No license handle supplied to an rlm_get_attr_xxx() call or rlm_license_xxx() call.
RLM_EL_SERVER_DOWN	-20	Server closed connection	The license server closed the connection to the application.
RLM_EL_NO_HEARTBEAT	-21	No heartbeat response received	Your application did not receive a response to a heartbeat message which it sent. This would happen when you call rlm_get_attr_health(), or automatically if you called rlm_auto_hb().
RLM_EL_ALLINUSE	-22	All licenses in use	All licenses are currently in use, and the user did not request to be queued. This request will succeed at some other time when some licenses are checked in.
RLM_EL_NOHOSTID	-23	No hostid on uncounted license	Uncounted licenses always require a hostid.
RLM_EL_TIMEDOUT	-24	License timed out by server	Your application did not send any heartbeats to the license server and the license administrator specified a TIMEOUT option in the ISV server options file.
RLM_EL_INQUEUE	-25	In queue for license	All licenses are in use, and the user requested queueing by setting the RLM_QUEUE environment variable.
RLM_EL_SYNTAX	-26	License syntax error	This is an internal error.
RLM_EL_ROAM_TOOLONG	-27	Roam time exceeds maximum	The roam time specified in a checkout request is longer than either the license-specified maximum roaming time or the license administrator's ROAM_MAX_DAYS option specification.
RLM_EL_NO_SERV_HANDLE	-28	Server does not know this license handle	This is an internal server error. It will be returned usually when you are attempting to return a roaming license early.
RLM_EL_ON_EXC_ROAM	-29	On roam exclude list	The license administrator has specified an EXCLUDE_ROAM list for this product, and the user (host, etc) is on it.
RLM_EL_NOT_INC_ROAM	-30	Not on the roam include list	The license administrator has specified an INCLUDE_ROAM list for this product, and you are not on it.
RLM_EL_TOOMANY_ROAMING	-31	Too many licenses roaming already	A request was made to roam a license, but there are too many licenses roaming already (set by the license administrator ROAM_MAX_COUNT option).
RLM_EL_WILL_EXPIRE	-32	License expires before roam period ends	A roaming license was requested, but the only license which can fulfill the request will expire before the roam period ends.
RLM_EL_ROAMFILEERR	-33	Problem with roam file	There was a problem writing the roam data file on the application's computer.

RLM_EL_RLM_ROAM_ERR	-34	Cannot check out rlm_roam license	A license was requested to roam, but the application cannot check out an rlm_roam license.
RLM_EL_WRONG_PLATFORM	-35	Wrong platform for client	The license specifies platforms=xxx, but the application is not running on one of these platforms.
RLM_EL_WRONG_TZ	-36	Wrong timezone for client	The license specifies an allowed timezone, but the application is running on a computer in a different timezone.
RLM_EL_NOT_STARTED	-37	License start date in the future	The start date in the license hasn't occurred yet, e.g., today you try to check out a license containing start=1-mar-2030.
RLM_EL_CANT_GET_DATE	-38	time() call failure	The *time()* system call failed
RLM_EL_OVERSOFT	-39	Request goes over license soft_limit	This license checkout causes the license usage to go over it's soft limit. The checkout is successful, but usage is now in the overdraft mode. RLM_EL_OVERSOFT is also returned if you have a misconfigured token-based license and the server has gone into overdraft due to this. See the note in the token-based license restrictions section.
RLM_EL_WINDBACK	-40	Clock setback detected	RLM has detected that the clock has been set back. This error will only happen on expiring licenses.
RLM_EL_BADPARAM	-41	Bad parameter to rlm_checkout() call	This currently happens if a checkout request is made for < 0 licenses.
RLM_EL_NOROAM_FAILOVER	-42	Roam operations not allowed on failover server	A failover server has taken over for a primary server, and a roaming license was requested. Roaming licenses can only be obtained from primary servers. Re-try the request later when the primary server is up.
RLM_EL_BADHOST	-43	bad hostname in license file or port@host	The hostname in the license file is not valid on this network.
RLM_EL_APP_INACTIVE	-44	Application is inactive	Your application is set to the inactive state (with rlm_set_active(rh, 0), and you have called rlm_get_attr_health().
RLM_EL_NOT_NAMED_USER	-45	User is not on the named-user list	You are not on the named user list for this product.
RLM_EL_TS_DISABLED	-46	Terminal server/remote desktop disabled	The only available license has Terminal Server disabled, and the application is running on a Windows Terminal Server machine.
RLM_EL_VM_DISABLED	-47	Running on Virtual Machines disabled	The only available license has virtual machines disabled, and the application is running on a virtual machine.
RLM_EL_PORTABLE_REMOVED	-48	Portable hostid removed	The license is locked to a portable hostid (dongle), and the hostid was removed after the license was acquired by the application.

RLM_EL_DEMOEXP	-49	Demo license has expired	Detached Demo[tm] license has expired.
RLM_EL_FAILED_BACK_UP	-50	Failed host back up - failover server released license	If you application is holding a license from a failover server, when the main server comes back up, the failover server will drop all the licenses it is serving, and you will get this status.
RLM_EL_SERVER_LOST_XFER	-51	Server lost it's transferred license	Your license was served by a server which had received transferred licenses from another license server. The originating license server may have gone down, in which case, your server will lose the licenses which were transferred to it.
RLM_EL_BAD_PASSWORD	-52	Incorrect password for product	RLM_EL_BAD_PASSWORD is an internal error and won't ever be returned to the client - if the license password is bad, the client will receive RLM_EL_NO_SERV_SUPP
RLM_EL_METER_NO_SERVER	-53	Metered licenses require server	Metered licenses only work with with a license server.
RLM_EL_METER_NOCOUNT	-54	Not enough count for meter	There is insufficient count in the meter for the requested operation.
RLM_EL_NOROAM_TRANSIENT	-55	Roaming not allowed	Roaming is not allowed on servers with transient hostids, ie, dongles.
RLM_EL_CANTRECONNECT	-56	Can't reconnect to server	On a disconnected handle, the operation requested needed to reconnect to the server, and this operation failed.
RLM_EL_NONE_CANROAM	-57	None of these licenses can roam	The license max_roam_count is set to 0. This will always be the case for licenses that are transferred to another server.
RLM_EH_SERVER_TOO_OLD	-58	Server too old for this operation	In v10, this error means that disconnected operation (rlm_init_disconn()) was attempted on a pre-v10.0 license server.
RLM_EH_SERVER_REJECT	-59	Server rejected client	Either the server is older than the oldest version allowed, or a generic server is used when the client specifies this is not allowed.
RLM_EL_REQ_OPT_MISSING	-60	Required option missing	A required option was set with the *rlm_set_req_opt()* call, and this string is not part of the license string. This error will only be reported for nodelocked licenses, the server will always report RLM_EL_NO_SERV_SUPP
RLM_EL_NO_DYNRES	-61	Reclaim can't find dynamic reservation	The license specified to be reclaimed cannot be found.
RLM_EL_RECONN_INFO_BAD	-62	Reconnection info invalid	This is generally an internal error.
RLM_EL_ALREADY_ROAMING	-63	License already roaming on this host	If you attempt to roam N licenses then later N+X licenses, you will receive this error. The original roam must be returned first.
RLM_EL_BAD_EXTEND_FMT	-64	Bad format for	RLM_ROAM_EXTEND format is

		RLM_ROAM_EXTEND	product:date:extension_code
RLM_EL_BAD_EXTEND_CODE	-65	Bad extend code	Extension code does not verify correctly.
RLM_EL_NO_ROAM_TO_EXTEND	-66	No roaming license to extend	This is an attempt to extend a non-existant roaming license.
RLM_EL_NESTED_ALIAS	-67	Nested aliases	You cannot define an alias in terms of another alias
RLM_EL_NO_JSON	-68	No JSON in returned message	This is an internal error in web services processing with RLMCloud
RLM_EL_BAD_JSON	-69	Bad JSON in returned message	This is an internal error in web services processing with RLMCloud
RLM_EL_BADHANDSHAKE	-70	Bad handshake on web services checkout	This is an internal error in web services processing with RLMCloud
RLM_EL_HELPER_ERR	-71	rlm_helper error	This is an internal error in web services processing with RLMCloud
RLM_EL_PERS_NOT_ON_LIST	-72	Not on list	The user is not on the personal license list
RLM_EL_PERS_BADPASS	-73	Bad password	The personal user's password is incorrect
RLM_EL_PERS_INUSE	-74	License in use	The personal license is in use
RLM_EL_PERS_HANDLE_ERR	-75	Handle error	RLM handle error (personal license)
RLM_EL_NOTTEMP	-76	Not a temp license	License is not a temporary license
RLM_EL_NOSERVER	-77	No server	No server to connect to return temp license

Table of Contents

Appendix C – RLM Example Client Program

This example program (rlmclient.c) is contained on the RLM kit in the *examples* directory.

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

            COPYRIGHT (c) 2005, 2009 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>
#ifndef _WIN32
#include <unistd.h>
#endif /* _WIN32 */

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

int
main(int argc, char *argv[])
{
  RLM_HANDLE rh;
  RLM_LICENSE lic;
  int stat, x;
  const char *product = "test1";
  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));
               exit(1);
        }
       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 wspecified, 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(0);
}

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));
        }
}
```

Table of Contents

Appendix D – Example rlm_isv_config()

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

            COPYRIGHT (c) 2005, 2019 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:    rlm_isv_config.c - configuration data for ISV
 *
 *      M. Christiano
 *      11/25/05
 *
 */

#include "license.h"
#include "license_to_run.h"

/*
 *      Define "INCLUDE_RLMID1" to include support for RLMID1 dongles.
 *      Comment out to remove aladdin dongle support.
 *
 *      Note: The RLMID1 dongle code is always included in
 *      your license server.  This setting is only for your applications, and
 *      only needs to be set if you are issuing licenses that are nodelocked
 *      to a dongle.
 *
 *      Including the RLMID1 dongle code increases the size of
 *      your applications by approx 900Kb on 32-bit windows, plus involves
 *      a small delay at application startup time, even if you are not using
 *      a dongle.
 *
 *      If you are not planning to issue licenses which are node-locked to
 *      rlmid devices, Reprise Software recommends leaving these options turned
 *      off (ie, leave the "#if 0" on the next several lines).
 */

#if 0
#define INCLUDE_RLMID1
#endif

#ifdef INCLUDE_RLMID1
extern void _rlm_gethostid_type1(RLM_HANDLE, L_HOSTID);
#endif

void
rlm_isv_config(RLM_HANDLE handle)
{

/*
 *      Set ISV name
 *
 *      NOTE: IF you are evaluating RLM, DO NOT change the ISV
 *            name, or your license keys will no longer work.
 *            For eval kits, the name on the next line MUST
 *            be "demo".
```

```
 *
 *      NOTE: Your ISV name is, in general, case-insensitive.
 *            The ONLY exception to this is when it is used as
 *            a lockfile name using a FLEXlm-compatible lockfile.
 *            In this case (and this case only), the case of the
 *            name you enter here is important.  Note that even in
 *            this case, ONLY THE LOCKFILE NAME uses the exact case
 *            you enter - every other place in RLM uses a lowercase
 *            version of this name.
 *
 *      Beginning in RLM v7.0, your ISV name is contained in
 *      "license_to_run.h".  If you need to alter the case of the
 *      name for a compatible FLEXlm lockfile, you should do it there
 *      and leave the next line as it is.
 *
 */
       rlm_isv_cfg_set_name(handle, RLM_ISV_NAME);


/*
 *      Set RLM license - do not modify this line
 */
       rlm_isv_cfg_set_license(handle, RLM_LICENSE_TO_RUN);


/*
 *      Set oldest allowed server version.
 *
 *      The next setting controls the oldest RLM license server
 *      version with which your application will work.
 *
 *      The 3 parameters are rlm version, revision, and build (in
 *      that order).
 *
 *      If you leave this set to 0, 0, 0, your application will
 *      attempt to work with the oldest available RLM server.
 *
 *      You should only set this if you are concerned with an older
 *      server in the field which has been hacked, otherwise, you should
 *      leave it set to 0, 0, 0.
 *
 *      (Note: Do not set this to anything between 0,0,0, and
 *       9,0,0).  Servers older than v9.0 will appear to be v0.0)
 *
 */
       rlm_isv_cfg_set_oldest_server(handle, 0, 0, 0);

/*
 *      Set ISV server settings file compatibility
 *
 *      The next setting controls what versions of RLM your
 *      ISV server settings file will work with.   You can enable
 *      it for all earlier versions (> v6), or later versions or both.
 *      The 2nd parameter enables earlier versions if non-zero, the
 *      3rd parameter enables later versions if non-zero.  Note that
 *      "earlier" and "later" are relative to the version of your
 *      settings file.  So, if you create the settings file with RLM v8,
 *      "earlier" means v6 and v7, while "later" means v9 and above.
 *
 *      default is: rlm_isv_cfg_set_compat(handle, 0, 1); - sets compatibility
 *                      with later versions, but not earlier ones.
 */
       rlm_isv_cfg_set_compat(handle, 0, 1);

/*
 *      Setup virtual machine enable/disable.
 *
 *      By default (if you do not modify the following call), RLM
```

```
 *      will refuse to run a license server on a virtual machine.
 *
 *      You can always enable a particular virtual machine by issuing
 *      an "rlm_server_enable_vm" license for that machine.
 *
 *      If you want license servers to run on all virtual machines, set
 *      the 2nd parameter of the next call to a non-zero value.
 *
 */
        rlm_isv_cfg_set_enable_vm(handle, 0);

/*
 *      Beginning in RLM v10.0, roaming is disabled for servers that
 *      use transient hostids (ie, dongles, or ISV-defined transient hostids).
 *      If you want to enable roaming on these servers, set the 2nd
 *      parameter of the next call to 1.
 */
        rlm_isv_cfg_set_enable_roam_transient(handle, 0);

/*
 *      Beginning in RLM v10.0,  you have the option of turning ROAMED
 *      licenses into "single" licenses.  Prior to RLM v10.0, all ROAMED
 *      licenses were nodelocked, uncounted.
 *      If you want your roamed licenses to be "single" licenses, set the
 *      second parameter of the next call to 1.
 */
        rlm_isv_cfg_set_roam_single(handle, 0);

/*
 *      FLEXlm(R)-style lockfile compatibility.
 *
 *      Set to non-zero to use a FLEXlm-style lockfile.  For windows
 *      systems, a value of 1 uses the 'C' drive always, whereas a
 *      value > 1 will use the system drive.  FLEXlm (up to version
 *      10.3, at least) puts the lockfile on the 'C' drive.
 *
 *      Reprise Software recommends setting this to 1 if you want to
 *      use FLEXlm-compatible lockfiles.
 */
        rlm_isv_cfg_set_use_flexlm_lockfile(handle, 0);

/*      The Windows disk serial number hostid code can return hostids
 *      that are usable only by processes running with admin rights if
 *      running with admin privileges.  If an application is installed
 *      and a license activated by an admin user, it's possible that
 *      a non-admin user will not be able to use the application because
 *      it can't read the disk serial number.  Beginning in RLM v10.0,
 *      you can disable the use of disk serial number hostids that are
 *      usable by admins only.  If you want to do so, change the second
 *      parameter of the next function to 0.
 */
#ifdef _WIN32
        rlm_isv_cfg_set_use_admin_disksns(handle, 1);
#endif

/*
 *      Beginning in RLM v10.0,  RLM's license transfer capability also
 *      allows for disconnected operation on the destination server.
 *      This capability only requires that an "rlm_roam" license be
 *      present on the destination server.  You can ship an rlm_roam
 *      license to your customer and have them install it on every
 *      destination server, or you can simply put it into the next
 *      call, in which case, no separate license file will be needed
 *      on the destination license server.
 *
 *      To enable this, set the 2nd parameter of the next call to a valid,
 *      signed rlm_roam license (enclosed in "<>") in place of the
 *      last argument.  This license should be a static string
```

```
 *      which is available for the lifetime of the server.
 *
 *      This license MUST have the following parameters:
 *              version: "1.0"
 *              exp: "permanent"
 *              count: "uncounted"
 *              hostid: "any"
 *              NO other parameters
 *
 *      for example:
 *
 *      rlm_isv_cfg_set_server_roam(handle, "<LICENSE your-isvname rlm_roam 1.0
uncounted hostid=any sig=xxxxxxx>");
 */
        rlm_isv_cfg_set_server_roam(handle, (char *) 0);

/*
 *      Beginning in RLM v10.0,  RLM can broadcast to find a license
 *      server as a last resort, if all the normal methods to find
 *      the server fail.  This capability is enabled by default.
 *
 *      To disable this, set the 2nd parameter of the next call to 1.
 */
        rlm_isv_cfg_disable_broadcast(handle, 0);

/*
 *      Beginning in RLM v11.0, the client can specify that
 *      it will not use a generic license server (i.e., rlm + a
 *      settings file).
 *      If you want to disable generic servers, set the 2nd
 *      parameter of the next call to 1.
 *      If you disable generic servers and your application
 *      attempts to connect to a generic server, it will
 *      receive an RLM_EH_SERVER_REJECT error upon connection
 *      or an RLM_EL_SERVER_REJECT upon license checkout.
 *      The text error message is "Server rejected client".
 *
 *      Pre-v11 clients will get a "Communications error with
 *      license server (-17), Connection refused at server (-111)"
 *      error in this case.
 */
        rlm_isv_cfg_disable_generic_server(handle, 0);

/*
 *      Beginning in RLM v10.1, licenses can be cached on the client
 *      side with the use of the "client_cache" license attribute.
 *      This capability must be enabled with the following call.
 *      If the 2nd parameter is 1, client caching is enabled, if 0,
 *      caching is disabled.
 */
        rlm_isv_cfg_enable_client_cache(handle, 1);

/*
 *      Beginning in RLM v10.1, license servers can return one
 *      valid license to the application which is then verified on
 *      the client side.  This check helps ensure that the license
 *      server hasn't been modified.  To enable this checking set
 *      the second parameter of the next call to 1.  If you enable
 *      this, please read the section titled "Server Integrity Checking"
 *      in the "Securing Your Application" section of the Reference
 *      Manual so that you understand the errors which can be generated
 *      as a result of this call and how you should proceed.
 */
        rlm_isv_cfg_enable_check_license(handle, 0);

/*
 *      Beginning in RLM v11.0, you can specify which types of
 *      hostids that Activation Pro will accept from an activation
```

```
 *      request.   Prior to v11.0, the only 6 types of acceptable
 *      hostids were: rehostable, isv-defined, rlmid, ethernet,
 *      disk serial numbers and native 32-bit hostids.
 *      In the following call, you can set the default hostids that
 *      your Actpro server will accept.  To get the pre-v11 behavior,
 *      set the 2nd parameter as shown.  Hostid type definitions in license.h
 *
 */
#if 0
{
  int allowed_types =   RLM_ACTPRO_ALLOW_REHOST | RLM_ACTPRO_ALLOW_ISV |
                        RLM_ACTPRO_ALLOW_ISVDEF | RLM_ACTPRO_ALLOW_RLMID |
                        RLM_ACTPRO_ALLOW_ETHER | RLM_ACTPRO_ALLOW_DISKSN |
                        RLM_ACTPRO_ALLOW_32 | RLM_ACTPRO_ALLOW_UUID |
                        RLM_ACTPRO_ALLOW_ASH;
        rlm_isv_cfg_actpro_allowed_hostids(handle, allowed_types);
}
#endif

/*
 *      Beginning in RLM v11.2, license servers can utilize
 *      Alternate Server Hostids.  These hostids are activated
 *      from Activation Pro by the ISV server, which needs to
 *      know the URL of the activation server.
 *      If you use Reprise's hosted activation service, the default
 *      (hostedactivation.com) is correct.  For all others, set your
 *      activation server url here.  Note that this URL pointer must
 *      remain valid as long as the RLM_HANDLE is in use.
 */
        /*** rlm_isv_cfg_set_url(handle, "hostedactivation.com"); ***/

/*
 *      Rehostable hostids do two checks at verification time which
 *      fail on certain systems.   These checks are:
 *         - checking the file ID of each file in the rehostable hierarchy, and
 *         - checking the native hostid of the system
 *
 *      The file ID check fails on Windows systems if drives are added or
 *      removed from the controller.
 *      We have seen the native hostid change on Centos systems when the
 *      network cable is unplugged.
 *
 *      Beginning in RLM v12.3, you can disable one or both of these
 *      checks by setting the second parameter of the two following
 *      calls to 1.  The default behavior remains the same as in
 *      previous versions of RLM.
 */
        rlm_isv_cfg_disable_windows_fileid_check(handle, 0);
                                        /* 0 -> check, <>0 -> no check */
        rlm_isv_cfg_disable_reference_hostid_check(handle, 0);
                                        /* 0 -> check, <>0 -> no check */
/*
 *      Roam extension is a new feature in RLM v12.3, and it is disabled
 *      by default.  If you enable it, be aware that the max_roam setting
 *      from your rlm_roam license will NOT be honored for a roam extension,
 *      only the max_roam setting of the license which is roaming.  This means
 *      that if you use max_roam on the rlm_roam license to limit roaming
 *      duration on your licenses, it will not be effective for any roam
 *      extension.  The default max_roam on any license is 30 days, so this
 *      may or may not be an issue for you.
 *
 *      To enable roam extensions, set the 2nd parameter of the next call
 *      to 1.  If you use a server settings file, you must re-generate the
 *      settings file with your v12.3 kit, otherwise, the roam extension will
 *      not appear in the RLM web interface.
 */
        rlm_isv_cfg_enable_roam_extend(handle, 0);
```

```
/*
 *      New in v12.4, the RLM web services API (used with RLMCloud) has an
 *      isv-defined server handshake function.  To use this, specify the 2
 *      parameters to the server-side of the algorighm here.  P1 is any
 *      32-bit number, but avoid long sequences of 0's or 1's.  P2 is any
 *      31-bit number, i.e. bit 31 (high-order bit) should be 0.
 *      NOTE: CHANGE THE DEFINTIONS of P1/P2 that appear here.
 */
#define P1 0x5c7daf39
#define P2 0x10030034
        rlm_isv_cfg_set_isv_handshake(handle, P1, P2);

/*
 *      Prior to RLM v12.4, if you enabled the check for server licenses
 *      by calling rlm_isv_cfg_enable_check_license(handle, 1), connections
 *      to the license server would fail if the license either contained new
 *      keywords or was invalid.  In v12.4 and later, you can cause the
 *      connection to succeed and retrieve the status later.  To do this,
 *      set the 2nd parameter of the next call to a non-zero value.   After
 *      connecting, you can call rlm_get_attr_checked_license() on the handle.
 *      rlm_get_attr_checked_license will return 0 for success or either
 *      RLM_NO_SERVER_LIC, RLM_LIC_NEW_KEYWORDS or RLM_LIC_BAD.
 */
        rlm_isv_cfg_no_server_license_fail(handle, 0);
                                        /* 0 -> check, <>0 -> no check */

/*
 *      To include RLMID1 dongle code, be sure INCLUDE_RLMID1 is defined above.
 */

#ifdef INCLUDE_RLMID1
        rlm_isv_cfg_set_use_hostid(handle, RLM_HOSTID_RLMID1,
                                                _rlm_gethostid_type1);
#endif
}
```

Table of Contents

Appendix E – 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	Platform Support	# of instances in RLM client	Notes
ANY	runs anywhere	ANY	all	1	
DEMO	runs anywhere for a demo license	DEMO	all	1	
serial number	runs anywhere	sn=123-456-789	all	1	used to identify a license, equivalent to string, any string up to 64 characters long
32	32-bit hostid, native on Unix, non X86 based platforms, System Drive Volume Serial Number on Windows	10ac0307	all	1	***should not be used on Linux or Mac***
disksn (see note below)	Disk hardware serial number	disksn=WD-WX60AC946860	windows	1	***Introduced in RLM v9.2***
ip (or internet)	TCP/IP address	ip=192.156.1.3	all	5	always printed as "ip=" (wildcards allowed starting in v3.0, see notes)
ether (See note below)	Ethernet MAC address	ether=00801935f2b5	windows, mac, linux	5	always printed without leading "ether="
rlmid1	External Key or Dongle	rlmid1=9a763f21	Windows, linux	6 (total of all rlmids)	USB Dongle. Always enabled in ISV license server. See Appendix F – Optional Hostid Installation Instructions on page 159 for details on building your software as well as extra software you need to ship with your product.
uuid	BIOS uuid	uuid=699A4D56-58BF- 1C83-D63C-27A8BEB8011A	Windows	1	
user	User name	USER=joe	all	1	case-insensitive
host	Host name	host=melody	all	1	Case-insensitive (wildcards allowed starting in v9.4 – see notes)

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 32
or
rlmutil rlmhostid internet

Note that rlmhostid will not return a string or serial number hostid type, since these values are unrelated to any particular computer - they are simply values that the ISV creates to differentiate licenses.

When an application requests a license from a license server, it will transmit the hostid information from

the local machine to the license server, so that the server can process node-locked licenses without additional queries to the application. The application will transmit a maximum of 25 different hostids:

- one 32-bit hostid, if present on this platform
- up to 5 IP addresses (ip=)
- up to 5 ethernet MAC addresses (ether=)
- up to 6 RLMID portable hostids
- a minimum of 3 ISV-defined hostids (usually more, but guaranteed to be at least 3)

A Note about Windows Ethernet hostids

Some interfaces on Windows systems have Ethernet MAC addresses which are undesirable for use as hostids because they are transient, i.e. not always available. These include wireless interfaces, virtual interfaces like VPNs, etc.

On Windows, RLM looks for keywords in the device description to decide what interfaces are undesirable. Licenses can be locked to these interfaces if necessary, as it might be that only undesirable interfaces exist on a given machine. However, When RLM generates a list of MAC addresses on a Windows machine, it orders the list such that the undesirables are at the end of the list. So the first hostid printed by rlmhostid, and the one returned by rlm_hostid() will be the best one available on that Windows system.

A Note about Linux Ethernet hostids

Some recent versions of linux have very high ethernet adapter numbers. RLM was updated to scan 100,000 interfaces starting in v12.0, however, this scan was too slow for applications that made multiple calls to rlm_init(). Beginning in v12.2, this number was reduced to 5,000. However, we have been informed of docker instances where the ethernet device number is between 14,000 and 15,000. In v15.0, we have left the default scan at 5,000 devices, but added the environment variable RLM_LINUX_ETHERNET_ITERATIONS to control how many devices are scanned. To scan more (or fewer) ethernet devices, set this variable as follows (this example sets the number to 20,000):

% setenv RLM_LINUX_ETHERNET_ITERATIONS 20000 (from the shell), or

rlm_putenv("RLM_LINUX_ETHERNET_ITERATIONS=20000"); in your application.

A Note about Windows disksn hostids

Some disk serial numbers on Windows are only accessible to a process running with admin privileges. To disable use of disk serial numbers that only admins can use, see the call to *rlm_isv_cfg_set_use_admin_disksns()* in rlm_isv_config.c".

Misc notes:

Note: The RLMID series of hostids are optional products, and will often require other software to be installed on the system on which they are to be used. For these devices, see Appendix F – Optional Hostid Installation Instructions, on page 159.

Note: Beginning in RLM v3.0, IP address hostids can contain the wildcard ('*') character in any position to indicate that any value is accepted in that position.

Note: Beginning in RLM v9.4, a wildcard may be used in the host type hostid, for example: "hostid=host=*.stanford.edu" or "hostid=host=*.reprisesoftware.com"

Disabling standard RLM Hostids

You can disable certain hostid types in your application in rlm_isv_config.c. *Note that rlmsign will sign licenses with disabled hostid types.* In your application, if this hostid type appears in a signed license file, rlm_checkout() will return RLM_EL_NOTTHISHOST. Your license server will log lines similar to these (in this case, we disabled the HOST hostid type, RLM_DISABLE_H_HOST):

> 06/14 14:44 (reprise) Wrong Hostid - licenses may not be available
>
> 06/14 14:44 (reprise) (expected: host=zippy, we are: invalid)

To disable hostids, modify rlm_isv_config.c as follows:

set the hostids that your product (or license server) will ***not*** accept. Create a bitmask of the hostid types you do ***not*** want to support, and pass this as the the 2nd parameter to rlm_isv_cfg_disable_hostids(). If this parameter is 0, all RLM hostids are allowed.

For example, say that you want to disable HOST and USER hostid types:

```
int disable = RLM_DISABLE_H_USER | RLM_DISABLE_H_HOST;
rlm_isv_cfg_disable_hostids(handle, disable);
```

The bitmask values for disabling various hostid types are in license.h, and are here:

- RLM_DISABLE_H_32BIT
- RLM_DISABLE_H_STRING
- RLM_DISABLE_H_ETHER
- RLM_DISABLE_H_USER
- RLM_DISABLE_H_HOST
- RLM_DISABLE_H_IP
- RLM_DISABLE_H_ANY
- RLM_DISABLE_H_DEMO
- RLM_DISABLE_H_SN
- RLM_DISABLE_H_RLMID1
- RLM_DISABLE_H_RLMID2
- RLM_DISABLE_H_GC
- RLM_DISABLE_H_DISKSN
- RLM_DISABLE_H_IPV6
- RLM_DISABLE_H_UUID

RLM Hostid Security

RLM hostids have varying levels of security. We describe these levels as:

- minimal (min) - the hostid works anywhere - nothing is required to run on any machine
- low - the hostid is locked, but the data it is locked to is easily changable, and in fact, the data is meant to be changed and changing it is fully documented. (in the case of Windows 32-bit hostids, which are the volume serial number, PC manufacturers often create batches of PCs with the same volume serial number).
- standard (std) - the hostid is locked to something which is not designed to be changed. Changing this requires some kind of hacking software, which may or may not be easily obtainable.

The following table shows RLM hostids and their security levels:

hostid type	security level	Notes
ANY	min	
DEMO	min	
32 (or long)	low or std	Depends on the platform, see table below
diskid	std	
gc	std	
ip (or internet)	low	
ether	std	
rlmid1	std	
user	min	
host	min	

The following table lists the security level of the 32-bit hostid type, by platform:

Platform	32-bit hostid security
hp_h1	std
hp64_h1	std
ibm_a1	std
ibm64_a1	std
x86_l1, x86_l2	low
ppc64_l1	low
x64_l1	low
x86_m1	low
x64_m1	low
ppc_m1	low
x64_s1	std
sun_s1	std
x64_s1	std
x86_w3/4	low
x64_w3/4	low

Table of Contents

Appendix F – Optional Hostid Installation Instructions

Certain hostids in the RLMID family (RLMID1) require device-specific installation on the target computer. These instructions must be passed on to your customer's license administrator in order for them to use the device. The RLMid1 device is a hardware key manufactured by Aladdin Knowledge Systems (now SafeNet, Inc).

Installing RLMid1 Devices on Windows

Installation on a target system can be accomplished in two ways:

- use Windows "Found New Hardware" to automatically load the drivers (preferred), or
- use the RLMID1 driver installer (from the Reprise Software website) to do the driver installation

Installation using the Found New Hardware Wizard

In order to use Windows to automatically do the driver installation, simply plug the device into the computer, and Windows will detect the new device. You will get the "Found New Hardware" wizard which will install the drivers for the "USB Protection Device" for you, as shown below:

Next, select "Install the Software Automatically (recommended)", and click "Next". Windows will locate the driver and install it. You will then get the "Completing the Found New Hardware Wizard" shown on the right; click "Finish".

That is all there is to it.

Installation using the driver installer.

If for some reason Windows fails to update the driver automatically, or if the target system is not connected to the internet, use the driver installer located at:
http://www.reprisesoftware.com/drivers/rlmid1.zip

you can also download the driver directly from the SafeNet site:

ftp://ftp.aladdin.com/pub/hasp/Sentinel_HASP/Runtime_%28Drivers%29/Sentinel_HASP_Runtime_setup.zip

Unzip the installer and run it on the target system.

Note that an RLMID1 device can be used by any RLM-licensed application on the system, in other words, there is nothing ISV-specific about the device.

Installing RLMid1 Devices on Linux

To install the necessary drivers for RLMid1 devices on linux, follow these steps:

1. Browse to http://sentinelcustomer.safenet-inc.com/sentineldownloads/. Look for the "Sentinal HASP/LDK Rutime Installer" for Linux. There are several options, depending on Linux variant and the style of installer you want (GUI, RPM, script).

2. Download the appropriate installer and install. Note that you will have to execute the installation as root.

   The runtime installer sets up a daemon that is used to access the hardware key.

Table of Contents

Appendix G - Release Notes

```
Release Notes - RLM v15.0BL1  May 23, 2022
_____________________________________________

This is the first production release of RLM v15.0.

v15.0BL1 is available on windows, mac, and linux systems.

------------------------------------------------------------------------
Notes on this release:

  - None
------------------------------------------------------------------------

************************************************************************

This release fixes bugs P567-P568, and P571-P572.

For each bug, we will indicate which RLM components need to be updated
for the bug fix.  This indication will be of the form:
        Fix requires: server
   or
        Fix requires: rlm, settings

This indication will list one or more of the following:

        - client - meaning you have to re-build your application.
        - rlm - meaning you need a new rlm binary (ie, you have nothing
                to re-build)
        - server - meaning you need a new rlm binary if you use the
                generic ISV server settings file, or a new ISV server
                if you use an ISV-specific server binary.
        - settings - meaning you need a new ISV server settings file.
        - actpro server - meaning you need a new activation pro server.


Known Issues in this release
_____________________________

For an up-to-date list of issues, see:
http://www.reprisesoftware.com/publisher/licensing-software-issues.php

Note:  The documentation is contained in 5 manuals:

  Standard RLM Components

    * RLM Getting Started Guide - an introduction to the basic concepts of
       license management and RLM (PDF)
    * RLM Reference Manual - the complete reference to all core RLM components
       (PDF)
    * RLM License Administration Manual - The stand-alone License
       Administration manual, suitable for shipment to your customers (Wiki)

  Optional RLM Components

    * RLM Activation Pro Getting Started Guide - an introduction to the RLM
       Activation Pro software (PDF)
    * RLM Activation Pro Manual - Reference for the Optional RLM Activation
       Pro software (PDF)

  All manuals are in PDF format, and are available on the Reprise Website at:
```

http://www.reprisesoftware.com/kits/RLM_Getting_Started_Guide.pdf
http://www.reprisesoftware.com/kits/RLM_Reference.pdf

http://www.reprisesoftware.com/kits/RLM_Activation_Pro_Getting_Started_Guide.pdf
http://www.reprisesoftware.com/kits/RLM_Activation_Pro.pdf
http://www.reprisesoftware.com/kits/RLM_License_Administration.pdf

What's new

(See the reference manual for complete descriptions)

- The RLM server will now determine the external IP address of the client and add this to the list of client IP hostids, if it is not already in the list. See â€œThe ISV Options Fileâ€ in the License License Administration Manual for more information.

- Prior to RLM v15.0, the linux ethernet hostid code would scan 5000 devices to get ethernet MAC addresses. In v15.0, the environment variable RLM_LINUX_ETHERNET_ITERATIONS will set the number of devices to check for MAC addresses. For more information, see A Note about Linux Ethernet hostids on page 273 for more information.

- The â€œDiagnosticsâ€ menu item in the RLM web interface now only appears if the user has â€œedit_optionsâ€, â€œedit_rlm_optionsâ€, â€œlogfilesâ€, or â€œallâ€ privs. See â€œRLM privileges assignable in the RLM password fileâ€ in the License Administration Manual for more information.

New License Keywords

- None

API additions

- None

API changes

- None

Options file changes

- The EXEMPT_MAX and EXEMPTALL_MAX ISV options have been added. See â€œThe ISV Options Fileâ€ in the License Administration Manual for more information.

Activation changes

- The Activation pro debuglog now includes the PID of the license generator process, so that it is easier to interpret the log in the case where multiple instances of the license generator are running simultaneously. See â€œDebugging Tabâ€ in the Activation Pro manual for more information.

- The â€œtext to prepend to licenseâ€ field has been increased from a maximum of 1024 to 30720 bytes. See â€œProduct Definitionsâ€ in the Activation Pro manual for more information.

```
Problems fixed in this release
_______________________________


This release fixes bugs P567-P568, and P571-P572.

P567 - If a failover server's primary license file contains an ASH hostid,
       the failover will core dump when taking over from the primary.

P568 - If activation with https is used, and there is a problem in the https
       certificate, the client can core dump.

P571 - if a server has floating and metered licenses checked out for one
       product, after a reread, all the metered clients are attached to the floating
       license in overdraft

P572 - A cloud license nodelocked to a hostid fails to show up in
       rlm_products() run on that hostid.  The same would be true with platform, tz,
       and computing_env


Platforms Supported
___________________

Linux on X86: Ubuntu 12.04.5 (x86_l2)
Linux on x64: Ubuntu 12.04.5 (x64_l1)
Linux on arm 64-bit: Ubuntu 1804 (arm64_l1)

Windows 32-bit - Visual Studio 2010 (x86_w3)
Windows 32-bit - Visual Studio 2015 (x86_w4)
Windows 64-bit - Visual Studio 2010 (x64_w3)
Windows 64-bit - Visual Studio 2015 (x64_w4)

Mac OS/X: 10.5.8 intel 64-bit (x64_m1)
Mac OS/X: 11.6.6 intel 64-bit (x64_m2)
Mac OS/X: 11.4 arm 64-bit (arm64_m2)


RLM Build environment
_____________________

x86_l2:
Linux 3.13.0-32-generic #57~precise1-Ubuntu SMP Tue Jul 15 03:50:54 UTC 2014 i686 i686
i386 GNU/Linux
gcc (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3

x64_l1:
Linux 3.13.0-32-generic #57~precise1-Ubuntu SMP Tue Jul 15 03:51:20 UTC 2014 x86_64 x86_64
x86_64 GNU/Linux
gcc (Ubuntu/Linaro 4.6.3-1ubuntu5) 4.6.3

arm64_l1:
Linux 5.3.0-1036-raspi2 #38-Ubuntu SMP Thu Oct 15 14:40:03 UTC 2020 aarch64 aarch64
aarch64 GNU/Linux
gcc version 7.5.0 (Ubuntu/Linaro 7.5.0-3ubuntu1~18.04)

x86_w3:
Windows XP Microsoft Visual Studio 2010

x86_w4:
Windows 7 Microsoft Visual Studio 2015

x64_w3
Windows XP Microsoft Visual Studio 2010

x64_w4
Windows 7 Microsoft Visual Studio 2015
```

```
x64_m1:
Darwin 9.8.0 Darwin Kernel Version 9.8.0: Wed Jul 15 16:55:01 PDT 2009; root:xnu-
1228.15.4~1/RELEASE_I386 i386
i686-apple-darwin9-gcc-4.0.1 (GCC) 4.0.1 (Apple Inc. build 5465)

x64_m2:
Darwin 20.6.0 Darwin Kernel Version 20.6.0: Wed Jun 23 00:26:31 PDT 2021; root:xnu-
7195.141.2~5/RELEASE_X86_64 x86_64
Apple clang version 12.0.5 (clang-1205.0.22.11)

arm64_m2:
Darwin 20.5.0 Darwin Kernel Version 20.5.0: Sat May  8 05:10:31 PDT 2021; root:xnu-
7195.121.3~9/RELEASE_ARM64_T8101 arm64
Apple clang version 12.0.5 (clang-1205.0.22.11)
```

Table of Contents

Appendix H - Frequently-Asked Questions

Reprise Software maintains a list of frequently-asked questions on our website. For the current list of Frequently-Asked Questions, please see our website.

For ISVs, see the FAQ at the reprise website at:

https://www.reprisesoftware.com/publisher/license-management-faq.php

For License Administrators, see the License Administrator FAQ at

https://www.reprisesoftware.com/admin/software-licensing-faq.php

Table of Contents

Appendix I – RLM Temporary Files

RLM stores rehostable hostids, etc in an "rlm directory" on your system.

The "rlm directory" is:

- /var/tmp on unix/mac (mac prior to RLM v14.1)
- /Library/Application Support/Reprise (mac for RLM v14.1 and later)
- ProgramData\Reprise (Windows vista and later)
- Docs and Settings\All Users\Application Data\Reprise (Windows pre-vista).

Files store here include:

- client-side detached demo files
- client-side “single” lockfiles
- client-side rehostable hostid data

The rehostable hostids are directories inside <rlm directory>/isvname

Notes on the v14.1 filename transition on mac

In the transition in v14.1, rlm will continue to look in old location (/var/tmp) for client-side “single” lockfiles. RLM will look in the new location for all other files listed above. meaning that detached demo/rehostable hostid definitions will not be preserved across the v14.1 RLM boundary. Rehostable hostids should be revoked on the old RLM version and re-activated in RLM v14.1.

Notes on the v14.1/v14.2 issues in rlm_init() on mac

In RLM v14.1, rlm_init() would return the error RLM_EH_NOTEMPDIR on MAC if the temporary directory was not present and could not be created. In v14.2 and later, this error is no longer returned. However, if the temporary directory can't be created, some other operations may fail later. These include any operations that utilize any of the following data:

- client-side detached demo files
- client-side “single” lockfiles
- client-side rehostable hostid data

If your application does not utilize any of the above data, you will not notice any unusual behavior without an RLM temporary directory on MAC. The license server still requires the temporary directory, and will refuse to run without one.

Table of Contents