RLM Reference Manual

Release 16.0.0

May 15, 2024

Welcome to the documentation for Reprise Software!

For questions, comments, or issues with the documentation, please contact support@reprisesoftware.com.

Chapter 1

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 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 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, Chapter 2, describes the various ways you can license your software with RLM.

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.

1.1 First, a few definitions

Table 1.1. Definitions

Terms	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 its 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.

1.2 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 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.

1.3 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
- start date

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

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

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

1.5 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 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.

Note 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.

2.1 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.

But your *License Model* can be (and often is) much more complicated. For example, you might sell a floating license which works only in certain time zones. You might also sell a more unrestricted floating license, which operates in any timezone. In this case, both *License Models* are of the basic floating type, but the other restrictions define the differences in the *License Models*. Your company might now call these "Floating-timezone-restricted" and "Floating-unrestricted".

Note 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 number of common License Models in the remainder of this chapter, and with each one, we will list the RLM license attributes that are used to implement it. Don't worry if you do not understand the RLM nomenclature, this will make more sense after you read the chapter describing Section 4.5.

2.2 Floating License

The floating license is what made license managers famous. All license managers support this. The idea is that some specified number of independent instances of the application can be run anywhere on your customer's network so long as that number does not exceed a predefined limit (the limit in the license file).

In RLM:

Every license has a **count**, and if the count is non-zero, this is a floating license for that many instances of the application. The license itself has no associated **hostid**, meaning that it will run anywhere. The license server (specified by the HOST and ISV lines) keeps track of the # of instances in use.

Example License

```
HOST serverName hostID 5053
ISV isvName
LICENSE isvName productName version# expiration count
```

2.3 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:

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.

Example License

```
LICENSE isvName productName versionNumber expiration count
hostid=XXXXXXXX
```

2.4 Node-locked, Limited License

A variant of the node-locked uncounted license, it is sometimes desirable to allow only a limited number of instances of the software to run on a particular computer. This is a counted license which is also nodelocked.

In RLM:

Set the **count** field of the license to a non-zero value (or **single**) and specify the **hostid** of

the computer in the actual license. This license requires a license server (HOST and ISV lines), unless you use a **single** license.

Example License

```
HOST localhost ANY 5053
ISV isvName
LICENSE isvName productName versionNumber expiration count
hostid=XXXXXXXX
```

2.5 Shared Floating License

Sometimes it is desirable to modify the behavior of a floating license so that all invocations of your product on a single computer use only one license. Or all invocations by the same user. Or all invocations from a particular process tree. In this case, license managers provide a method of sharing a particular license among multiple instances of the product.

In RLM:

Use a floating license and specify the **share=** attribute in the actual license.

Attribute	Result
share=u	Will cause all invocations from a **particular user** to share the license.
share=uh	Will cause all invocations from a **particular user** on a **particular machine** to share the license.
:count (optional)	Limits the number of shared copies before an additional license is used. (e.g. **share=u:4** allows 4 instances on a single license)

This license requires a license server (HOST and ISV lines).

Example License

```
HOST localhost ANY 5053
ISV isvName
LICENSE isvName productName versionNumber expiration count
hostid=XXXXXXXX share=uh
```

2.6 Named-User License

A named-user license allows a limited number of users access to a floating license. This allows you to sell a number of instances of your software to a subset of users at the customer site, without having to identify the users at the time you create the license.

In RLM:

Use a floating license and specify the **named_user** or **named_user=n** attribute in the actual license. If you specify **named_user**, then your customer can assign as many users as there are licenses available. Specifying **named_user=n** will allow you to set a lower (or higher) limit for the number of users.

This license requires a license server.

Example License

```
HOST localhost ANY 5053
ISV isvName
LICENSE isvName productName versionNumber expiration count
hostid=XXXXXXXX named_user=3
```

2.7 Metered License

A metered license allows you to allocate usage and consume the allocation as something happens in your software. For example, you could meter the number of times your program is run (as opposed to the *concurrent limit of execution* in a floating license). Or you could count the number of pages printed in a word-processing application. Metering also allows you to create a license which will run for a predetermined amount of running time (program running time, as opposed to an expiration date – expiration dates are usually available for all license types).

In RLM:

Set the **count** field of the license to **meter**, and specify the metering parameters in the actual license. The metering parameters control how usage is consumed when you call **rlm_checkout()** and subsequently as the application continues to run.

This license requires a license server (HOST and ISV lines).

Example License

```
LICENSE isvName productName versionNumber expiration meter
hostid=XXXXXXXX
```

2.8 Token-Based License

A token-based license defines the license you request in terms of other licenses (called the primary license or licenses) which were issued to your customer. This can be used for several different purposes:

- Allows you to define several licenses in terms of a single, common primary license. Your customer can purchase many copies of the primary license, then they are allowed to run whatever products are defined to use that license. A big advantage is as you release new products which use the same primary license, your customer can use these products immediately, creating more contention for the licenses, and additional sales for you.
- Allows you to create product *bundles* or *packages*.
- Allows you to provide a mapping from a particular license request to one or several equivalent licenses. Typically, ISVs are happy to allow more expensive licenses to be used to satisfy the request of a lower-cost product. This allows your customers to keep working while increasing contention for the higher-priced licenses.

In RLM:

Set the **count** field of the license to **token**, and specify the primary license(s) which are used to satisfy the request for the product. The token license is generally the same for all your customers, and you issue licenses of the **primary** license when they purchase your software. This license requires a license server.

Example License

```
HOST localhost ANY 5053
ISV isvName
LICENSE isvName productName versionNumber expiration token
hostid=XXXXXXXX
```

2.9 "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:

Set the **version** field of the license to a **date**, in the format **yyyy.mm**, and specify the version in your call to rlm_checkout() in the same release date 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.

Warning

While it is possible to use other date formats, the format above is used by
RLM Activation Pro.

Example License

```
HOST localhost ANY 5053
ISV isvName
LICENSE isvName productName 1999.01 expiration count hostid=XXXXXXXX
```

What's New in RLM 16.0

Important Version 16.0 brings an all-new web management interface to RLM, with many changes. End-user documentation may need to be updated before distributing RLM to your users. For an overview of the new interface, see the overview video on our blog.

- **The RLM web interface now uses HTTPS by default.** Self-signed certificates are automatically generated on startup. If no cert or key are found, the web server will be disabled. See "enabling-https for more information.
- **The RLM web server now starts with a default login and password.** Users will be prompted to change password on first login.
  - Default login: admin
  - Default password: admin
- **New user permissions**: View, Manage, and Admin
- **Diagnostics are now presented directly in the web interface.** Users now have the option to download directly to their local machine.
- ISV actions now available via the kebab (**?**) menu on each server.
- License and usage information now directly available on home page.
- RLM and ISV options files are now read-only from the interface.

Important The rlm.pw file is no longer used. User credentials are now managed within the web server.

3.1 Problems fixed in this release

- **Bug**: Activation URL incorrectly shortened when using RLM web UI to activate a license
- **Bug**: Cookie issues with Safari browser
- CVE-2018-15574
- CVE-2021-37500
- CVE-2021-44154
- CVE-2021-45422
- CVE-2022-28363
- CVE-2022-28364
- CVE-2022-30519

- CVE-2023-43183
- CVE-2023-44031

3.2 Known issues in this release

- Users may experience slow loading times in the web management interface when running RLM from a Windows machine. This is fixed by installing the Sentinel HASP drivers on the server machine.

The following functionality is not available and will be added in 16.1:

- Metered licenses
- Alternate Server Hostid
- License Transfer

3.3 Platform changes

No changes

3.4 Activation changes

No changes

3.5 Dependency changes

No changes

RLM Basics

4.1 Welcome

4.1.1 Integrating RLM into your product

As an ISV you integrate RLM by adding calls from the RLM 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 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, the additional components you ship are:

If you provide only Fixed (node-locked) licenses to your customers:

- 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.

Note NO SERVERS NEEDED

If you provide Concurrent-use (floating) licenses to your customers:

- 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.
- The rlm (generic) license server provided by Reprise Software.
- Your custom license server (Built from components from Reprise Software with a minimum of configuration)

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 concurrent-use (floating) licenses, RLM is a client-server system, with license requests transmitted over TCP/IP from the client (your application) to the license server that controls license usage rights. When deployed to support node-locked licensing, no network connection nor license server processes are needed.

In all cases, all components, except the license file, 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.

4.1.2 What sets RLM apart?

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

RLM is **open** because we publish the format of our report log file, so that you, or your license administrators can always examine and generate usage reports on licensing activity from the RLM servers.

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 include functionality like truly automatic selection of license servers from a set of multiple, independent servers. In older license management systems, the ISV ends up writing much code to manage multiple license servers. This is handled by RLM itself.

4.2 Installing RLM

To install RLM, follow these steps:

4.2.1 First, Download the kit from the Reprise website

To download RLM, go to the Reprise Website Download area, enter your username and password, and select the kit(s) you want to download. Save these on your system, then uncompress and (on Unix) extract the binaries with the *tar xvf* command.

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 ".zip" (Windows) appended.

View Platform Kits

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 on Intel X86 (Visual C 2013)	x86_w3	rlm.vX.YBLZ-x86_w3.zip
Windows 64-bit on Intel X86 (Visual C 2013)	x64_w3	rlm.vX.YBLZ-x64_w3.zip
Windows on Intel X86 (visual C 2015+)	x86_w4	rlm.vX.YBLZ-x86_w3.zip
Windows 64-bit on Intel X86 (visual C 2015+)	x64_w4	rlm.vX.YBLZ-x64_w3.zip
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

4.2.2 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 Section 5.2.
- For information on using RLM in a cross-development environment, see Section 4.2.4.

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
```

Warning RLM requires a license to operate from Reprise Software.

On Windows, the kit is a Windows .zip file. Extract the .zip file, whose name is rlm.*ver-platform*.zip, where *ver* is the RLM version and platform is the RLM platform name. For example, rlm.v13.0BL1-x86_w4.zip is the kit for v13.0BL2 on the x86_w4 platform. Extract the .zip file to the preferred installation location (for example: C:\Reprise\).

Previously, the Windows kit came packaged as an executable that provided the option to copy the key pair from an existing RLM installation into the new install. The key pair must now be manually copied if there is an existing installation. Copying the key pair 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. Copying the key pair is important for having compatible license signatures across the installations and is recommended.

RLM kits are pre-built for ISV "demo", with licenses that expire 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 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.

Warning 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 Command Prompt".

2. cd to the platform directory of the SDK, for example:

```
cd x86_w3
```

3. Type nmake

4.2.3 A note about OpenSSL

Note As of RLM 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.

4.2.4 Building the RLM 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.

4.2.5 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”.

Java support is contained in an independent directory called either “java_unix” or “java_win”.

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)

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_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	
NetBSD on Intel	x86_n1	
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 2013
Windows 32-bit	x86_w4	Visual Studio 2015 and later
Windows 64-bit	x64_w3	Visual Studio 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_isv_config.c	Configuration data for ISV server.
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.
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.
unsupported	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	Not on client-only kits
rlm.a	RLM library	
rlmanon	RLM logfile anonymizer	Not on client-only kits
rlmmains.a	RLM main() functions for misc programs	
rlmutil	RLM utilities	

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

File	Contents
example.lic	Example license file.
isv_main.obj	main() for ISV server.
isv_server.lib	library for ISV server.
makefile	Makefile
rlc.obj	main() for Activation administration (rlc).
rlm.def	RLM DLL export definitions.
rlm.exe	The generic RLM server.
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 anonymizer.
rlmclient.lib	RLM client library.
rlmclient_md.lib	RLM client library - compiled with /Md.
rlmclient_mdd.lib	RLM client library - compiled with /Mdd.
rlmclient_mtd.lib	RLM client library - compiled with /Mtd.
rlmgen.obj	rlc license generation module.
rlmgenkeys.obj	main() for rlmgenkeys utility.
rlmsign.obj	main() for rlmsign utility.
rlmutil.exe	RLM utilities.
rlmverify.obj	main() for RLM log file authentication utility.
x86_w*.vcproj, x64_w*.vcproj	Visual Studio/Visual C++ project for building the SDK.

The Java directory (java_unix, java_win) contains:

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

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

File	Contents
Reprise	Visual Studio 2022 Project Directory for RLM .NET support.
RLMTest	Visual Studio 2022 Project Directory for RLM .NET Test program.

4.3 Integrating RLM Into Your Product

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

1. A client library
2. A license server (RLM has 2 license servers - a generic server called *rlm* and an ISV-specific server.)
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 license server is used for floating licenses and logging of usage data. Your license administrators also have the ability to control certain aspects of the license server's operations by specifying options in The ISV Options File in the License Administration manual.

The RLM client library (linked into your application) and the license server are both 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 the license servers 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 the license server, 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 the license server performs in a more standard way 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.

4.3.2 INTEGRATING RLM 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 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.

4.3.3 1. Decide on your Licensing Strategy

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

The most basic license attributes are:

- ISV name (you pick this when you purchase RLM)
- Product name
- Highest Version supported
- Node-locked or floating (if node-locked, the node identification)
- Expiration date

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

Note 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.

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

Before you use RLM, 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 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.

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

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

4.3.5 3. Add RLM API calls to your application

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

These functions are described in Section 6.1.

4.3.6 Core API

These 8 functions provide all basic licensing operations needed for most applications:

Function	Description
rlm_init(), rlm_init_disconn()	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 by checking server.

If you have special licensing needs that are not addressed by these functions, see Section 6.1 which lists all RLM API functions.

4.3.7 4. Configure and build your RLM Kit

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

1. Install your RLM license.
2. Create your public/private key pair, which is done one time only and which was done in

step #2, above. (See Section 4.3.4).

3. Configure your RLM parameters.
4. 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 license, first retrieve the license from our activation server using the activation key given to you when you purchased RLM (keep this key handy, it will not change across versions of RLM or when you buy new platforms). Next, cut and paste the output into the file src/license_to_run.h

---

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).

Expand for an example license_to_run.h.

```
(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:

1. Your Public Key, for license key verification - rlm_pubkey.c - (This was done in step #2, above. See Section 4.3.4).
2. A file of RLM customizations called rlm_isv_config.c (this file is contained in the src directory on the kit)

rlm_pubkey.c is created by the rlmgenkeys utility. **You should run this only once** to create your public/private key pair. Once you create these files, save them - if you lose one of these files,

you will no longer be able to generate license keys compatible with older versions of your software.

4.3.8 Customizing RLM with rlm_isv_config

rlm_isv_config.c contains calls to:

- set up your ISV name
- install your RLM license (do not change this call)
- specify the oldest server your application can use
- specify the range of servers your settings file works with
- enable the server to run on virtual machines
- enable roaming on servers that use transient hostids
- create "single" licenses when licenses are roamed, instead of uncounted
- disable the RLM clock windback detection for expiring licenses
- create a FLEXlm-compatible lock file
- enable or disable Windows disk serial numbers which require admin access to use
- enable disconnected license transfer
- enable or disable client-side broadcast to find the license server
- disable the use of the generic license server
- enable license client-side caching
- enable license server security checks
- 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)
- disable the file id check and native hostid check for rehostable hostids
- enable the remote time extension of roaming licenses.
- Define isv-defined encryption handshake for the rlm web services API
- specify the promise interval for HTTPS communications.
- Specify hostid types that are disabled.

Edit this file before compiling your isv server, license generator, or applications.

Note Your ISV name is, in general, case insensitive. The *ONLY EXCEPTION* to this rule is in the case of FLEXlm-compatible lock files. Since FLEXlm uses case-sensitive ISV names, the lockfile name must be case-sensitive. Therefore, if you are creating a FLEXlm-compatible lockfile, you should enter your ISV name in the exact case as in FLEXlm. The case of the name will affect the lockfile name, but only the lockfile name. Everywhere else in RLM, your isv name will be converted to lowercase.

Once you have created these 2 files you create your ISV server by typing "make" in the kit directory, and you are ready to link your applications with the RLM libraries.

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).

4.3.9 5. Package your software for shipment

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

- Review the RLM API calls you make in your application to be sure that you use product names that are suitable (we strongly recommend using the name of the product that is in general use), and that the version numbers are correct. If you intend for your customers to be able to use old licenses from your product, be sure that the version number in the *rlm_checkout()* call is appropriate.
- If we have provided you with special debug libraries, make sure you use the nondebug libraries from the standard kit for your release.
- Review the options you used to Build Your License Server.
- Ensure that you have included the RLM Server, your ISV Server, and the RLM 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 Section 6.4 for more details.
- Review the Best Practices for RLM Integration section and ensure that your product and installation are well-behaved.

4.3.10 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 Section 4.5.

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 Section 4.6.

License creation tool

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 other license parameters. For more information on rlmsign, see Section 4.6.

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 Section 6.1.

Note Do not call *rlm_sign_license()* in an application or utility that ships to customers. Doing so will cause your private key to be included in the application executable or binary, which could expose it to hackers, possibly enabling them to create counterfeit licenses for your product.

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, the following Product Names are in use:

Product Names

Name	Description
rlm_demo	Used by RLM to enable **Detached Demo:sup:`TM`** licenses for your products.
rlm_failover, *rlm_failover_server*	Used by RLM to enable failover license servers on a customer-by-customer basis.
rlm_roam	Used by RLM to enable license roaming for your products.
rlm_server	Used by RLM to create alternate hostids for license servers. Also note that the *rlm_server* license will not be visible in status requests, or in *rlm_products()* calls.
rlm_server_enable_vm	Used by RLM to enable license servers to operate on a particular virtual machine. (Note that you can enable your server to work on all virtual machines by calling rlm_isv_cfg_set_enable_vm() with the second parameter set to a non-0 value.) Also note that the *rlm_server_enable_vm* license will not be visible in status requests, or in *rlm_products()* calls as of RLM v9.0.
rlm_no_server_lock	Used to configure your ISV server so that multiple copies can be run on a single system. This should be done for in-house use of your ISV server only, for example, when supporting cloud computing with a number of your customers. Note that the *rlm_no_server_lock* license will not be visible in status requests or in *rlm_products()* calls. The reason this is for in-house use only is because if license administrators could run multiple copies of your ISV server he could serve the same license file from each copy, thus multiplying the number of licenses you intend for him to run.

Note 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.

4.3.11 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:

1. In a command window, build the RLM SDK as specified in Installing RLM. You need do this only once per release of RLM.
2. **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
        - comsupp.lib
        - ole32.lib
        - oleaut32.lib
        - libcmt.lib

    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.

4.3.12 Using optional RLMID hostids

RLM currently supports one optional hostid choice:

- **rlmid1** - a hardware key manufactured by now SafeNet, Inc.

This optional hostid has specific requirements both when you build your product and when you ship it; it is available only on certain platforms.

Your license server (and the generic server) will be pre-built to support all rlmid devices on all supported platforms, so you are already enabled to use rlmid1 hardware keys as a server hostid. This means that if you only want to use rlmid1 devices to lock your license server, there is nothing more to do.

In your application, however, you need to enable the various rlmid hostids should you chose to use the particular hostid for nodelocked licenses.

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
x86_w4	v12.0
x64_w2	v5
x64_w3	v9
x64_w4	v12.0

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 support, license locked to ANY	37 seconds	55.55	0.018
rlmid1 support, license locked to rlmid1	40 seconds	50	0.02

rlmid1

The rlmid1 devices are manufactured by 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, and support is configured into the ISV server by default - no additional installation steps are required. However, 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
```

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 attempts to determine the ID of any rlmid1 devices connected to the local system.

For information on shipping your product with *rlmid1* devices, see Section 6.4.

4.3.13 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 Section 6.1 for a description of all the available RLM API functions.

4.3.14 Clock Tampering Detection

RLM 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 affect this check.

4.3.15 RLM Host IDs

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

Table 4.1. RLM's host identification (hostid) types are:

Type	Meaning	Example	Notes
ANY	runs anywhere	hostid=ANY	
DEMO	runs anywhere for a demo license	hostid=DEMO	
serial number	runs anywhere	hostid=sn=123-456-789	Used to identify a license, any string up to 64 characters long.
disksn	Hard Disk hardware serial number	hostid=disksn=WDWX60AG94686O	Windows only.
32	32-bit hostid, native on Unix, non X86 based platforms	hostid=10ac0307	This is the volume serial number on windows, and is not recommended.
ip (or internet)	TCP/IP address	hostid=ip=192.156.1.3	Always printed as "ip=". Wildcards allowed.
ether	Ethernet MAC address	hostid=ether=00801935f2b5	Always printed without leading "ether=".
rlmid1	External key or dongle	hostid=rlmid1=9a763f21	External key or dongle.
uuid	BIOS uuid	hostid=uuid=699A4D56-5BB1-6C83-DE3C-27A8BEB8011A	Windows only.
user	User name	hostid=USER=joe	Case-insensitive
host	Host name	hostid=host=melody	

gc	*Google Compute Engine*	*hostid=gc=3797742226458986650-.k6qt9v5h38w2adwqgc9fdhdf3w0m761p*	*Linux only, DEPRECATED.*

Warning The Google Compute host ID is deprecated as of v14.2 and should not be used.

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
```

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 * 1 or 2 Disk serial numbers (windows only) * up to 3 IP address * up to 5 ethernet MAC addresses (ether=) * up to 6 RLMID portable hostids * a UUID hostid, if present * a minimum of 3 ISV-defined hostids (usually more, but guaranteed to be at least 3)

Note In RLM v15.0, the server will add the client's external IP address to the list of IP addresses supplied by the client. This IP address is not added if it is the same as one of the client-supplied IP addresses.

For more information see Section 6.3.

4.4 Best Practices for Integrating RLM

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. 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.

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

- Licenses present in your product's binary directory.
- Options you provide to your user to specify a license location.
- 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.
- If you are not using an ISV-defined hostid , **ship the ISV server settings file** rather than the ISV server binary. Using the settings file means that your server-side executables are completely generic, and your customers can upgrade RLM versions and get bug fixes via a download from Reprise, and you have no ISV server build-test-release cycle to go through.
- **Include a folder for licenses in your installed product folder tree.** In this folder, if you ship floating licenses, you would create a license file with a single HOST line similar to the following:

```
HOST myserver any 5053
```

Note This HOST line does not need a valid hostid, only a correct port# and hostname). At runtime, your application passes the path to this directory as the first argument to rlm_init(). Then any license you ever issue - an expiring demo license, a production nodelocked license, or a license file simply containing a HOST line as described above - 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.

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.

4.5 The License File

The license file contains information which configures the license servers and describes all the licenses granted from the ISV to your customer. There are 4 basic license types:

- **counted**
- **uncounted**
- **token**
- **metered**

Everything else in the license file is a modification of one of these 4 basic license types. For example, if a counted license has a hostid associated with it, it becomes a **nodelocked, counted license**. If the counted license has no hostid, it is a **floating license**. The various attributes modify the basic license types. For example, if a counted license has a **share=** attribute, then multiple application instances can use a single license. If the license file has counted or metered licenses, it requires a license server, so a HOST and ISV line must be present. If the license file has only uncounted licenses, the HOST and ISV lines are not required.

License Files have 6 types of lines:

Line	Description
HOST	Specify the license server host (SERVER is an alias for HOST – see Section 4.5.3)
ISV	Specify the ISV's license server information (DAEMON is an alias for ISV – see Section 4.5.4)
LICENSE	Describe license grants from the ISV to your customer (FEATURE is an alias for LICENSE – see Section 4.5.5)
UPGRADE	Upgrade the version number of some or all LICENSEs, (see Section 4.5.7)
CUSTOMER	**RLM Cloud only** - See: Section 4.5.8
Comment	(see next section)

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

Note RLM does not support byte order markers (BOMs) in license files. If a BOM is present in a license file, RLM will not recognize the content.

4.5.1 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 Prior to RLM 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.

The product name *rlm_roam* is treated specially by RLM. *rlm_roam* indicates that roaming has been enabled by an ISV. If an ISV issues an *rlm_roam* license, then roaming is enabled on any computer which is able to check out the *rlm_roam* license while in a disconnected state.

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 except as noted below: "<", ">", "&", single quote ('), back-quote (`) and double-quote (").

As of RLM v12.2, "<", ">", "&", single quote (') and back quote (`) are legal characters in the customer field.

ISV license names cannot begin with the characters "rlm_".

Note All lines in license files as well as option files (RLM or ISV) **must** be shorter than 1024 characters. Anything over 1024 characters will be truncated.

Everything in the license file is case-insensitive, with the following three exceptions:

- *isv-binary-pathname* on ISV lines **[Case-sensitive on Unix systems only]**
- *options-file-filename* on ISV lines **[Case-sensitive on Unix systems only]**
- Short (~62-character) license keys (keys with bits/character of 6 - see Section 4.6)

Note Any time RLM processes a *username*, it will replace any white space in the name with the underscore '_' character. This is true for *usernames* used as hostids, in server option files, or passed between client and server. Also note that *usernames* are case-insensitive – they are converted to all lowercase in the license server.

The following sections describe each of the three types of license file lines (Section 4.5.3, Section 4.5.4, Section 4.5.5).

4.5.2 Order of lines in the license file

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

- For an ISV/DAEMON line which specifies a password, all LICENSE/FEATURE lines following the ISV/DAEMON line which do not specify a password will use the password of the ISV/DAEMON line which precedes it. If the ISV/DAEMON line follows the LICENSE/FEATURE line, the password will not apply to that LICENSE/FEATURE.
- 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.

4.5.3 HOST Line

```
HOST hostname hostid [tcp/ip port #]
```

The HOST line specifies which computer the license server is to run on. There is only one HOST line per license file. Note that if a license file contains only nodelocked-uncounted licenses, a HOST line is not required.

The *hostname* is the standard TCP/IP hostname for the system. This name is not an input to the license key signature algorithm, and thus can be changed at any time.

The *hostid* is RLM's idea of the computer's identification. The *hostid* is an input to the license key signature algorithm, so it cannot be changed. All licenses in the license file have this *hostid* as input to their license signatures, so it is important to avoid moving LICENSE lines from one license file to another, as this invalidates them.

The tcp/ip port number is the port which RLM attempts to use for communications. This number can be changed to any available port.

For a description of the various hostids that RLM supports, see Section 4.3.15.

Example:

```
HOST melody 80b918aa 2700
```

In this example, the license servers run on host "melody" at TCP/IP port # 2700.

Note The keyword "SERVER" can be used instead of "HOST".

4.5.4 ISV Line

Format:

```
ISV isvname [isv-binary-pathname [options-file-filename [port-number]]]
[binary=isv-binarypathname]
[options=options-file-filename] [port=port-number] [password=password-text]
```

The ISV line specifies an ISV's license server. There is one ISV line in the license file for every *isvname* which has licenses in that file. Note that if a license file contains only nodelocked-uncounted licenses for a particular ISV, an ISV line is not required for that ISV.

Parameter	Required	Description
`isvname`	Yes	The name assigned to the ISV - **does not change**.
`isv-binary-pathname`	Yes*	Location of the ISVs license server binary. Can be any accessible location. Not an input to license key signature algorithm so can be changed any time. * *Can be omitted if ISV server is in same location as the RLM binary.*
`options=options-file-filename`	No	Specifies whether an options file will be used (see: isv-options-file). Specify options file location here, or name the ISV options file *isvname*.opt and place in same directory as license file.
`port=port-number`	No	Specifies port ISV server will use. If omitted, port is allocated dynamically.
`password=password-text`	No	Specifies license password applied to all LICENSE or FEATURE lines which follow the ISV line in the license file. If individual LICENSE/FEATURE line has a password, password from LICENSE/FEATURE line is used.

Any of the optional parameters can be specified by themselves. Also note that any number of the positional parameters can be specified, and optional parameters can be added after the positional parameters.

Note If you specify the same parameter both as a positional parameter and as a keyword=value parameter, the value of the keyword=value parameter will be used.

Examples:

License server for ISV *reprise* is located at /home/reprise/reprise, options file is located at /home/reprise/reprise.opt:

```
ISV reprise options=/home/reprise/reprise.opt binary=/home/reprise/reprise
```

License server for ISV *reprise* located at /home/reprise/reprise and ISV server port # specified:

```
ISV reprise /home/reprise/reprise port=8765
```

License server for ISV *reprise* located at /home/reprise/reprise and ISV server binary name /a/b/reprise used instead of /home/reprise/reprise:

```
ISV reprise /home/reprise/reprise binary=/a/b/reprise
```

Note The keyword "VENDOR" can be used instead of "ISV".

4.5.5 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, i.e., 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 The license file parser will reject unknown keywords.

Fixed (positional) parameters

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

isv

The name of the ISV granting the rights.

product

The name of the product for which license rights are being granted.

version

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 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

The date the license expires, in the form dd-mmm-yyyy, for example, 1-jul-2007. All licenses have an 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). The license expires at midnight on the date specified, in other words, it is valid for the entire day of the expiration date.

You can also specify an expiration time, using the Expiration time keyword.

The license expiration date can be either **dd-mmm-yyyy** (e.g., 01-jan-2023), or it can be formatted entirely numerically as **yyyy-mm-dd** (e.g., 2023-01-01).

RLM uses a proprietary algorithm to check for clock windback. This algorithm does not access any other computers and has been used in RLM since version 1.0. It is fast but sometimes returns false positives.

count

The number of licenses granted. The count field defines the license type. See the Chapter 2 for a discussion of license types and modifiers. The license type is one of:

- A positive integer indicates a **counted license**.

- 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, node-locked 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.
- token, token_bound, and token_unlocked are used to specify a **Token-Based License**; this license must also have the token=… optional parameter (see Section 5.5). The only optional parameter on a token-based license which is used by RLM is the start date. All other optional parameters are ignored.
- Meter indicates a **metered license**. See Section 5.4 for a complete description of metered licenses.

license-key

This 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 If the license-key is preceded by sig=, it can be present after any or all of the optional parameters.

Licenses can also have the following optional license modifier attributes:

Locking: Node-locked, Username-locked, or Floating licenses.

RLM can lock a license in a variety of ways:
node-locked (counted or uncounted)

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, see Section 4.3.15.

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 counted, uncounted, or "single". If it is uncounted or single, then the software need only verify that it is executing on the correct computer, and no license server is required. If it is counted, however, a license server is required to maintain a count of licenses currently in use.

Note While uncounted nodelocked licenses can be served by a license server, single licenses cannot.)

To create a node-locked license, add the keyword **hostid=..** at the end of the license line.
username-locked (counted or uncounted)

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 Any white space in a username is converted to the underscore ('_') character. Also note that usernames are case-insensitive.

---

floating

A license can be floating. This license will work anywhere on the network that can communi-

catewith the license server. To specify a floating license, do not put a hostid= keyword on the license.

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 supports, see Section 4.3.15.

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"
```

Other optional license parameters

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. The maximum length of the activation key is 40 characters, including the "akey=" part.

Caching Licenses on the Client Node

client_cache=seconds

When specified, the license will be held by the license server for "seconds" seconds, in exactly the same way as a "minimum_checkout" attribute would. However, on the client side, license data will be cached which can be used by subsequent checkout requests. The maximum cache time is one hour (3600 seconds). Note that the client-side caching will be ineffective if not enabled with rlm_isv_cfg_enable_client_cache() in rlm_isv_config.c, however the server will still hold the license checked-out in this case.

You can think of this as a very short-term, automatic, roam.

You should note that only licenses which have a sharing attribute will be usable as cached licenses. In other words, if the license has a "share=u" attribute, and the same user attempts a checkout, the cached license can be used. If the "share=" attribute is missing, no checkouts of the cached license will be possible. Also note that the host attribute is matched automatically, by definition, since the license is being used on the same computer.

Your customer can modify the value of client_cache between 0 and 2x the value you specify with the license administration CLIENT_CACHE option. See CLIENT_CACHE in the License Administration manual for more details.

Disable Computing Environments

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. Put the list in quotes if more than one item is specified.

Environment	Description
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 either for the client (for nodelocked licenses) or on the server (for floating licenses).

Hold time and Minimum Checkout

hold=N and **min_checkout=n**

By specifying *hold=N* in the license, the license will be held for N seconds after the application exits or checks the license back in via rlm_checkin(). hold is typically used in the case of licenses that have a very short duty-cycle, in order to provide a "fairer" measure of concurrent usage.

min_checkout specifies that a license is to be "held" checked-out by the license server after the application performs a check-in call or exits, only if the license did not remain checked out for the minimum interval specified by *n*. n must be a positive integer, greater than 0. The license will remain checked-out such that the total checkout time is n seconds. *min_checkout* is typically used in the case of licenses that have a very short duty-cycle, in order to provide a "fairer" measure of concurrent usage.

hold and *min_checkout* both perform this function in slightly different ways. hold always keeps the license checked out for the specified amount of time, whereas *min_checkout* keeps the license checked out for an additional time only if the license was checked back in by the application before the specified minimum time.

Note Both *hold* and *min_checkout* time specifications are ignored for any license which is roaming. Also note that hold time is processed by the license server, so it has no effect on unserved nodelocked licenses.

License ID

_id=nnn

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 ID# (assuming all the other attributes make it eligible to pool).

In addition, beginning in RLM v12.0 licenses are sorted (within a license file) in the order of the _id keyword. Licenses without _id keywords will remain unsorted (in their original order) at the end of all the licenses with _id keywords, which are sorted in increasing numerical order. This sort is done prior to REPLACE processing.

Other than license pooling and sorting, 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

issued=dd-mmm-yyyy

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.

Maximum Roam Count

max_roam_count=num

A Roaming license is a license that is checked out from the license server and used on a disconnected system. During this time, the license server holds the license checked-out just as if the system were still connected to the license server. Roaming licenses are only allowed if you issue your customer an *rlm_roam* license that is valid on the disconnected system.

If you do not specify *max_roam_count* in an individual license, RLM allows the total number of licenses to be roamed. Setting *max_roam_count* to a number less than the total number of licenses will cause the server to only allow that number of licenses to roam. To disable roaming on a particular license, set *max_roam_count=0* for that license.

Note The ISV-supplied *max_roam_count* attribute is equivalent to the license administrator MAX_ROAM_COUNT option, except that *max_roam_count* in the license is always enforced, rather than being optional.

Maximum Roam Days

max_roam=days

If you do not specify *max_roam* in an individual license, RLM limits the maximum number of days that a license can roam to 30 days. To disable roaming on a particular license, set *max_roam=-1* for that license.

Specifying *max_roam* on the *rlm_roam* license itself will potentially lower the max_roam of all products. The effective *max_roam* is the minimum of the value specified in the license itself (or the default value) and the value in the rlm_roam license. So, for example, if you set *max_roam=20* on the rlm_roam license, then all licenses without a max_roam will be limited to 20 days. If you set *max_roam=40* on the rlm_roam license, then only individual licenses with max_roam set to **greater** than 40 days will be affected.

Metered licenses

Metered licenses are described in detail in Section 5.4. Metered licenses use the following 4 keywords:

- **meter_counter=counter#**
- **meter_dec=decrement_amount**
- **meter_period=period_in_minutes**
- **meter_period_dec=periodic_decrement_amount**

Named User licenses

named_user[=num] or **named_user="num min_hours"**

Named User Licenses allow the ISV to require that a list of users be created who can use

the license(s). The number of users in the list can be less than, equal to, or greater than the number of licenses available. Once a user is added to the list, they can be deleted, but once deleted, they must remain off the list for a minimum number of hours (24 hours by default).

To create a **named user** license, add the **named_user** keyword to the LICENSE:

named_user	Require the same # of users as there are licenses.
named_user=n	Require *n* users to be named.
named_user="n min_hours"	Require *n* users, and specify the minimum number of hours.

With a **named_user** license, the license server can construct the list of users automatically as license checkouts occur, or the list can be entered via the RLM web interface by the license administrator. If entered manually, either individual users or GROUP names (as defined in the ISV server options file) can be used.

named_user licenses utilize the INCLUDE functionality of the license server, and do not need the entire list of **num** users specified before the licenses can be used. In fact, no users need to be specified since the license server will add users who do not appear on the list if the current list size is less than the number of allowed named users.

Once a user is added to the list, they can be removed at any time. However, once removed, a user cannot be placed back on the list until they have been off the list for **min_hours** hours (default: 24 hours).

Note

- Different named_user licenses **will never be combined** into one license pool, even if all other license parameters match.
- named_user utilizes the INCLUDE list in the server, so **any INCLUDE specification for this license will be ignored**.
- named_user is processed by the license server, so it **has no effect on unserved nodelocked licenses**.
- **usernames are case-insensitive**, so user "Joe" is the same as user "joe".

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 If the string contains embedded white space, it must be enclosed within double quotes.

You can specify a required substring in the options field with the *rlm_set_attr_req_opt()* call. See rlm_set_attr_req_opt() for more information.

License Password

_password = password-string

A license password restricts access to this license to requests which have specified the same password-string. The password-string is specified either with the *rlm_set_attr_password()* call, or (for the command-line utilities) with the RLM_LICENSE_PASSWORD environment variable, or in the RLM web interface. The password string must be <= 32 characters.

Note The license password does not factor into the license signature, and hence can be changed at any time after the license is signed. Also note that license passwords only work with served licenses, not nodelocked, uncounted or single licenses in local license files.

If a request (checkout or status) is made for a license which contains a password, and the request does not specify a matching password, the license server will return RLM_EL_NO_SERV_SUPP (for a checkout request) or no data (for a status request). This prevents a user from knowing that a license exists and attempting to guess the password.

The license password can be specified on the ISV line with the password=password-text optional field. If specified on the ISV line, the password applies to all LICENSE or FEATURE lines which follow the ISV line in the license file. If any individual LICENSE/FEATURE line specifies a password, the password from the LICENSE/FEATURE line is used.

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 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, 32-bit	hp_h1	hp_h
HP//UX Itanium, 64-bit	ia64_h1	ia64_h
IBM AIX, 32-bit	ibm_a1	ibm_a
IBM AIX, 64-bit	ibm64_a1	ibm64_a
IBM Power Linux, 64-bit	p64_l1	p64_l
Linux Intel, 64-bit	x64_l1	x64_l
Linux ARM, 64-bit	arm64_l1	arm64_l
Linux, 32-bit	x86_l2	x86_l
Linux, 64-bit	x64_l1	x64_l
Linux Itanium, 64-bit	ia64_l2	ia64_l
Linux PPC, 64-bit	ppc64_l1	ppc64_l
Mac Arm, 64-bit	arm64_m2	arm64_m
Mac Intel, 32-bit	x86_m1	x86_m
Mac Intel, 64-bit	x64_m2	x64_m
Mac PPC, 64-bit	ppc_m1	ppc_m
NetBSD, 32-bit	x86_n1	x86_n
Solaris Sparc , 32-bit	sun_s1	sun_s
Solaris Sparc, 64-bit	sun64_s1	sun64_s
Solaris, 64-bit	x64_s1	x64_s
Windows Intel, 32-bit	x86_w3, x86_w4	x86_w
Windows Intel, 64-bit	x64_w3, 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). 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.

Warning If two replace licenses have the same issued/start date, **they will both be active** after the replace determination has been made.

Effective Start Date

start=dd-mmm-yyyy

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

License Soft Limits.

soft_limit=n

A license can have a *soft_limit* that is lower than its count of available licenses. Once usage exceeds the *soft_limit*, checkout requests will return the RLM_EL_OVERSOFT status instead of a 0 status.

Note When the license server pools separate licenses into a single license pool, the *soft_limit* of each license is added to obtain the pool's soft_limit. Licenses which do not specify a soft_limit use the license count as their soft_limit.

Soft limits are processed by the license server, so they have no effect on unserved nodelocked licenses.

Sharing of licenses between different processes

share=UHI[:nnn]

Licenses can be shared between separate running processes by using a *share*= specification in the license. A license can be shared between processes with the same **username**, **hostname**, or **ISV-defined** data; or any combination of these. Specify share as *share=UHI* where the keywords 'U', 'H', and 'I' indicate that the Username, the Hostname, or the ISV-defined fields must match. **If more than one is specified, all specified must match in order to share the license**.

Note Usernames and hostnames in RLM are case-insensitive.

The share= keyword accepts an optional maximum process count which can share the license.

To specify a maximum process count for a license that is shared by user, use **share=U:nnn**, where *nnn* is the number of processes which can share this license. The nnn+1st request will consume an additional license.

If the :nnn specification is omitted, any number of processes can share the license.

Note Once a shared license is in use, it will continue to be in use until all the processes sharing the license check it back in. In other words, if 2 processes are sharing one license, and a 3rd process consumes a 2nd license (in the case where n==2), 2 licenses will continue to be in use until either (a) the 3rd process checks in its license, or (b) BOTH the first and second processes check in their licenses. In other words, there is no dynamic re-combination of shared licenses done at license check-in time.

Note Share is processed by the license server, so it has no effect on unserved nodelocked licenses.

Table 4.2. Examples

share=UH	Both username and hostname of a request must match.
share=U	Only the usernames must match on two processes to share license.
share=U:2	With matching usernames, one license is consumed for every two shares.

User, Host-based or Personal licenses.

user_based[=n] host_based[=n] personal (RLM Cloud only)

User-based licenses allow the ISV to require that the specified number of licenses (or all licenses) must be reserved to users (with RESERVE lines) in the options file. Note that the special user '*' does not count as being reserved. If fewer than the required number of licenses are reserved, the license server will log an error and discard the license. Also note that licenses reserved to a GROUP are not counted, otherwise the license administrator could simply bypass the intent of this license by creating a GROUP consisting of all their users. Thus, all reservations must be to individual users.

Similarly, host-based licenses allow the ISV to require that the specified number of licenses (or all licenses) must be reserved to hosts (with RESERVE lines) in the options file. Note that the special host '*' does not count as being reserved. If fewer than the required number of licenses are reserved, the license server will log an error and discard the license. Also note that licenses reserved to a HOST_GROUP are not counted, otherwise the license administrator could simply bypass the intent of this license by creating a HOST_GROUP consisting of all their hosts. Thus, all reservations must be to individual hosts.

Personal licenses (which are only available on RLM Cloud servers) allow the ISV to require that the authorized users be preassigned with the RLM Cloud portal GUI.

To create either user-based, host-based, or personal licenses, add the appropriate keyword to the LICENSE:

user_based[=nnn]	For user-based licenses.
host_based[=nnn]	For host-based licenses.
personal	For personal licenses.

If the optional "=nnn" is specified, only "nnn" of the total number of licenses need to be reserved, otherwise all licenses must be reserved.

For personal licenses, no count is specified, the personal user count is the license count.

Note user_based and host_based are processed by the license server, so they have no effect on unserved nodelocked licenses.

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

Minimum rlmremove interval

min_remove=n

min_remove specifies the lowest value, in seconds, a License Administrator can set the MINREMOVE value for a license.

If not specified in the license, the RLM default of 120 seconds (2 minutes) is used.

If min_remove is set to -1, then rlmremove (and the Remove button in the RLM web interface) is disabled on this license.

Note min_remove is processed by the license server, so it has no effect on unserved nodelocked licenses.

Minimum Timeout specification

min_timeout=n

A license administrator can specify a TIMEOUT value for any idle license. If the license remains idle (i.e.. does not communicate with the license server) for this amount of time, the license server performs an automatic check-in of the license and informs the application (if it is still running).

min_timeout=n specifies the lowest value a license administrator can set for the timeout value for a license. If not specified in the license, the RLM default minimum of 3600 seconds (1 hour) is used.

Note Timeout is processed by the license server, so it has no effect on unserved nodelocked licenses.

Additional Fields

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"
- "subscription"

(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.)

Table 4.3. 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
hold	positive integer - seconds	2^{31} - 1
host_based	int	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
max_roam	non-negative integer - days	2^{31} - 1
max_roam_count	non-negative integer - count	2^{31} - 1
min_checkout	positive integer - seconds	2^{31} - 1
min_remove	integer - seconds (-1 for no remove available)	2^{31}
min_timeout	positive integer - seconds	2^{31} - 1
options	string	64 characters

password	string	32 characters
personal	int	2^{31} - 1
platform	string	80 characters
share	enumerated	3 (“uhi”) + :integer
soft_limit	int	2^{31} - 1
start	string of the form dd-mmm-yyyy	11 characters
timezone	int	bitmap with bits 0-23 set
user_based	int	2^{31} - 1
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” “beta” and/or “subscription”	14 characters

4.5.6 Valid LICENSE Option Configurations

Only certain combinations of license types and options are valid. The following table indicates which options are valid for various license types. An X in a cell indicates that the attribute on the left is valid with the main license type in the column; empty cells indicate unsupported option configurations. All license administrator editable options (beginning with “_”) can be added to any license type.

License Type →	Floating	Nodelocked	Single	Uncounted	Token	Metered
Attributes ↓						
HOST/ISV line	required	required	ignored	X	X	required
UPGRADE-able	X	X	X	X		
contract	X	X	X	X	X	X
customer	X	X	X	X	X	X
disable	X	X	X	X	X	X
hold	X	X		when served		
host_based	X					
hostid		required	required	required	X	X
issued	X	X	X	X	X	X
issuer	X	X	X	X	X	X
max_roam	X					
max_roam_count	X					
min_checkout	X	X				
min_timeout	X	X			X	X
named_user	X	X				
options	X	X	X	X	X	X
platforms	X	X	X	X	X	X
replace	X	X	X	X	X	X
start	X	X	X	X	X	X
soft_limit	X	X				
share	X	X				
timezone	X	X	X	X	X	X
type	X	X	X	X	X	X
user_based	X	X				

Examples

```
LICENSE reprise write 1.0 permanent uncounted 98743597834597345734987539874587345987345
hostid=IP=172.16.7.3
```

```
LICENSE reprise calc 1.0 1-aug-2008 5
9874359783459734573498753987495873459873345
```

In the first 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. The second example is for 5 licenses of product calc which expire on the first of August 2008. The first license would not require a HOST line, whereas the second one would.

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

Note Licenses are always additive, in other words, the counts on 2 license lines of the same product/isv/version/hostid would be added together by the license server and the total number of licenses would be available.

4.5.7 UPGRADE Line

Format:

```
UPGRADE isv product from-version to-version exp-date count [sig=]upgrade-key
[optional
parameters]
```

The UPGRADE line defines an upgrade from an older version (*from-version* or higher) to a newer version (*to-version*) of an existing *product*. All fields in the upgrade line are case-insensitive (with the exception of short, i.e., 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 an "_" character.

We refer to the license specified by the UPGRADE line as the *upgrade license*, and the license it operates on as the *base license*.

An UPGRADE license will convert *count* licenses of *product* of version *from-version* or higher into *to-version*.

Note An UPGRADE license will never operate on a base license that is >= to-version.

In order for the upgrade to be performed, certain parameters of the *upgrade license* must match the parameters of a *base license* in order for that license to be eligible to be **upgraded**.

Certain licenses can never be upgraded. In particular, token-based licenses (the token definitions themselves) will never be upgraded. Also, named-user and metered licenses are not eligible for upgrades.

In order for a license to be upgraded, the following parameters must match on the *base license* and the *upgrade license*:

- License Disable specification
- License Options
- License Sharing specification (share and max_share)
- License Timezone specification
- License Platform list

- Both licenses must have the same counting type: *counted, uncounted, or single*
- License node-locked hostid
- Both licenses must be user-based or host-based (or neither)

Note This list is the same as the criteria for a license server combining multiple licenses into a license pool.

If the license is eligible for an upgrade, *count* licenses of the base license will be transformed into *to-version* licenses. An **upgrade** can be performed on multiple *base* licenses, until the *count* on the *upgrade license* is exhausted.

Note License "replace" processing is done (both in client and server) before upgrade processing. This means that an upgrade license should not specify replacement of the base license which it is going to upgrade, because the base license will no longer exist when the upgrade is done.

There are 3 upgrade cases:

- Fewer *base licenses* than *upgrade licenses* - in this case, the extra *upgrade licenses* are "wasted", and the license server issues a warning. All base licenses are upgraded.
- Same number of *base licenses* and *upgrade licenses* - all base licenses are upgraded.
- Fewer *upgrade licenses* than *base licenses* - in this case, *count* licenses are upgraded, and the remaining *base licenses* remain at their old version.

The 3rd case above is the most useful - if you are upgrading all instances of an existing license, a *replace* option on a new license will do this just as well. The only advantage to the *upgrade license* in the first two cases is that the *base license* is required, i.e., the *upgrade license*, by itself, grants no rights.

When a license is **upgraded** by the license server, the new licenses will have their parameters modified as follows. All other license parameters will be the same as the *base license*:

Field	**Result for Served *counted (or uncounted)* licenses**	**Result for Unserved *single or uncounted licenses***
exp-date	earlier date is used	earlier date is used
hold	maximum of the 2 values	• undefined -
max_roam	minimum of the 2 values	• undefined -
min_checkout	maximum of the 2 values	• undefined -
min_remove	maximum of the 2 values	• undefined -
min_timeout	maximum of the 2 values	• undefined -
soft_limit (upgrading all)	Larger of 2 deltas from license count is used.	• undefined -
soft_limit (partial upgrade)	Upgrade soft limit is preserved for the new version, base license soft limit delta is preserved (minimum value 0) for the old version.	• undefined -

user_based, host_based (upgrading all)	If licenses have a specification, the value closest to the license count is used .	• undefined -
user_based, host_based (partial upgrade)	If licenses have a specification, upgrade spec is preserved for the new version; base license spec delta (countuser_based) is preserved (minimum value 1) for the old version.	• undefined -
contract	If base license is empty, use upgrade license. On partial upgrade, if upgrade license is empty, use value from base license for the new version.	If base license is empty, use upgrade license.
customer	If base license is empty, use upgrade license. On partial upgrade, if upgrade license is empty, use value from base license for the new version.	If base license is empty, use upgrade license.
issuer	If base license is empty, use upgrade license. On partial upgrade, if upgrade license is empty, use value from base license for the new version.	If base license is empty, use upgrade license.
type	If base license is empty, use upgrade license. On partial upgrade, if upgrade license is empty, use value from base license for the new version.	If base license is empty, use upgrade license.

Fixed (positional) parameters

The first 7 parameters are required on every upgrade line and are present in the order shown above.

isv

The name of the ISV granting the rights.

product

The name of the product for which license rights are being upgraded.

from-version

The lowest-numbered product version which is eligible for an **upgrade**, in the form "N.M".

For example, 1.0, 2.37, or 2006.12

to-version

The highest-numbered product version supported by the license once it is **upgraded**, in the form "N.M".

For example, 1.0, 2.37, or 2006.12

exp-date

The date the upgrade expires, in the form dd-mmm-yyyy, for example, 1-jul-2007. A non-expiring upgrade can be specified with either a year of "0" (i.e., "1-jan-0"), or simply the word "permanent".

count

The number of licenses to be upgraded. **0** or **uncounted** means an uncounted, node-locked license is to be upgraded. **uncounted and 0 are 100% equivalent**.

> **single** means a node-locked, single-use license is to be upgraded. single is different from 1. See Section 4.5.5 above for more information.
>
> **token**, token_locked, token_bound and token_unlocked are not allowed in an UPGRADE license. All other optional parameters are ignored.

Warning The keyword token_locked was deprecated in v12.4 and replaced with

token_bound. Licenses with token_locked keywords will continue to operate, but they will not be locked to the server's hostid).

upgrade-key

A digital signature of all the upgrade data, along with the hostid on the HOST line, if present. If an upgrade license has a non-zero count, it always requires a HOST line. An upgrade to 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 *upgrade-key*. The *upgrade-key* will have "sig=" prepended after the license has been signed by the *rlmsign* utility.

Note If the upgrade-key is preceded by sig=, it can be present after any or all of the optional parameters.

Optional Parameters

Optional parameters are sometimes present in a license and can be present in any order. Optional parameters are allowed only once in an UPGRADE line. The syntax and meaning of optional parameters for the UPGRADE line are identical to the same parameters for the LICENSE line. Note that **token** and **named_user** are not allowed on an UPGRADE line. See Section 4.5.5 for information on the optional parameters.

Example:

```
LICENSE reprise write 1.0 permanent 5 sig=.....
UPGRADE reprise write 1.0 2.0 1-aug-2015 5 sig=.....
```

In this example, the 5 licenses of the write product have been upgraded from v1.0 to v2.0.

Note This license file would require a HOST line, since the licenses are counted.

4.5.8 CUSTOMER line

```
CUSTOMER customer-name isv=isvname server=server-name port=port#
password=yourpw
```

The CUSTOMER line, used for the client side of RLM Cloud-served licenses only, is used to identify the correct license server for a particular customer.

customer-name

Your customer's assigned Customer name.

isvname

Your ISV name.

server-name

The fully qualified hostname of the RLM Cloud license server machine.

port#

Either 5053 for normal RLM communications, or for HTTPS transport, 443).

yourpw

Your assigned password for this server.

Note HTTPS support must be enabled by your ISV to be effective. Please consult with your ISV

to see if HTTPS support is available.

Example:

```
CUSTOMER yourco isv=reprise server=ls423.rlmcloud.com port=5053 password=xyz
```

isvname

Your ISV name.

4.5.9 The License Environment

All software that uses RLM attempts to read a license file or communicate with a license server. The specification of the license file or the license server is done with the license environment.

If the software is ISV-specific (e.g., an application program), the first place that is checked is any license file or port@host specified in the environment variable *ISV*_LICENSE (if it is defined), where ISV is name of the ISV (e.g., REPRISE_LICENSE).

If *ISV_LICENSE* is not defined (in the case of ISV software), or for generic software (RLM utilities, the RLM server) the path specified by the environment variable RLM_LICENSE is checked, if RLM_LICENSE is defined.

If neither environment variable is defined, the program's default location for the license is checked next. In the case of the RLM utilities and the RLM server, this is any file ending in *.lic* in the current directory. Note that the RLM utilities will substitute the path specification from a *-c* option in place of the current directory.

Finally, any *.lic* file in the directory containing the binary will be checked.

The format of the environment variable (RLM_LICENSE or *ISV*_LICENSE) is:

```
license_spec1
```

or

Listing 4.1. Unix

```
license_spec1:license_spec2:license_spec3: .... :license_specN
```

Listing 4.2. Windows

```
license_spec1;license_spec2;license_spec3; .... ;license_specN
```

where:

license_spec is a license specification of the form:

- port@host - or -
- license_file_pathname - or -
- directory pathname (containing license files, all of which are added to the list) - or -
- <actual text of license>

Note The separator character is ':' for Unix systems and ';' for Windows systems.

Note The license file cannot be placed in a path where any component of the pathname contains the '@' character.

Example

On Unix/Mac:

```
% setenv RLM_LICENSE 1700@myhost:/home/reprise/reprise.lic
```

On Windows:

```
C:\ set RLM_LICENSE=1700@myhost:/home/reprise/reprise.lic
```

In this example, the server at port 1700 on host "myhost" is checked first, then if the request is not satisfied, the license file at /home/reprise/reprise.lic is used. Note that this search order may be modified if the environment variable RLM_PATH_RANDOMIZE is set.

Example 2

```
% setenv RLM_LICENSE 1700@myhost:<LICENSE isv prod1 1.0 1-jan-09 uncounted
hostid=any
key="1234...">
```

This example specifies the license for "prod1" directly in the environment variable, in addition to using the server at port 1700 at host "myhost".

On Windows, you can also set the environment variable via the control panel using these steps:

1. Go to Control Panel->System (or Control Panel->System and Security)
2. Click on "Advanced System Settings"
3. Click on "Environment Variables"
4. Enter RLM_LICENSE as the environment variable name and the license file directory as the value

---

Note Beginning in RLM v14.2, white space is trimmed from the start and end of the license environment variable value.

Note Beginning in RLM v15.1, uppercase ISV_LICENSE can be used as well as isv_LICENSE on case-sensitive file systems. In the case that both are defined, the uppercase ISV_LICENSE will be used.

4.6 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.

4.6.1 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 Section 4.5.

Note Every time you generate an RLM license, it gets a unique signature. Since RLM detects "duplicate" licenses by comparing the license signature, a newly-generated license will appear to be unique. This means that if you re-generate a license for your customer, even if it has

the same parameters, it will add incremental licenses. If you need to regenerate licenses, Reprise Software strongly recommends that you use the issue date and the replace attribute.

4.6.2 License creation tools

There are 3 ways to create licenses using RLM:

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

Note There was previously a fourth way—rlmgen—that has since been deprecated.

4.6.3 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 its first parameter. rlmsign will sign the *license file* and make it ready to ship to your customer.

rlmsign reads the *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.

rlmsign returns a 0 status if licenses were signed successfully, or if there were no licenses to sign. Otherwise, rlmsign returns a standard rlm error code from license.h.

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]
```

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.

Table 4.4. *bits-per-character*

bits-per-character	Result
4	License key is hexadecimal and approximately 92 characters.
5	License key is numbers and uppercase letters only; the key is approximately 74 characters.
6	License is upper and lowercase letters, numbers, and 4 special characters (*, =, +, ~). Key is approximately 62 characters.

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.

4.6.4 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 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 Section 6.1.

Note Do not call *rlm_sign_license()* in an application or utility that ships to customers. Doing so will cause your private key to be included in the application executable or binary, which could expose it to hackers, possibly enabling them to create counterfeit licenses for your product.

4.6.5 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.

4.6.6 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. The Reprise Product Names currently in use are:

- **rlm_demo** - This product name is used by RLM to enable Detached Demotm licenses for your products.
- **rlm_failover**, **rlm_failover_server** - This product name is used by RLM to enable failover license servers on a customer-by-customer basis.
- **rlm_roam** - This product name is used by RLM to enable license roaming for your products.
- **rlm_server** - This product name is used by RLM to create alternate hostids for license servers. The rlm_server license will not be visible in status requests, or in rlm_products() calls.
- **rlm_server_enable_vm** - This product name is used by RLM to enable license servers to operate on a particular virtual machine. (Note that you can enable your server to work on all virtual machines by calling rlm_isv_cfg_set_enable_vm() with the second parameter set to a non-0 value.) Also note that the rlm_server_enable_vm license will not be visible in

status requests, or in rlm_products() calls.

4.7 The License Server

The license server consists of at least two processes:

- The generic server, called *rlm*
- At least one ISV server, named *isv*

The *RLM* server is provided by Reprise Software and is completely generic. The ISV server is configured to contain license key validation that is ISV-specific. The ISV server configuration is contained in a platform-independent **settings** (*isv*.set) file, rather than a different binary built for each platform.

The *RLM* server handles requests from clients and directs them to the appropriate ISV server. In addition, the *RLM* server watches for failures in ISV servers and restarts them when appropriate. The RLM server also provides status to the various utilities, when appropriate.

The *RLM* server initiates a reread of the license files (for itself and any ISV servers) at midnight every night.

The RLM server is delivered with an Section 4.7.1 to perform normal administration tasks.

The ISV server consists of either a **platform-specific binary** file (*isvname.exe* on Windows, or *isvname* on Unix), or a **platform-independent settings** file (isvname.set on either Windows or Unix). When you run the "make" command in your kit, both versions of the ISV server are created. Reprise Software recommends shipping the settings file, which is platform independent, however the older-style executable is built for you as well should you wish to ship this version instead. If you ship the settings file, not only is it platform-independent, but it is also optionally **automatically upgradable** in the field (this is the default setting in rlm_isv_config.c) - meaning that if your license administrator upgrades to a newer version of RLM (the RLM server), your ISV server will automatically be upgraded as well.

Warning If you use ISV-defined hostid code, you cannot make use of the Generic ISV server - you must build an ISV server binary on each of your supported server platforms.

4.7.1 RLM Embedded Web Server

The *RLM* server contains an embedded *Web Server* which can be used to perform most administration of the RLM server itself. The web server contains the functionality of all the rlmXXXX utilities except rlmhostid. The web server allows you to retrieve server and license status (similar to *rlmstat*), cause the servers to reread the license files (*rlmreread*), switch debug (*rlmswitch*) or report log (*rlmswitchr*) files, move the current report log file to a new name (*rlmnewlog*), or shut down the license servers (*rlmdown*). Using this web-based interface, you can administer the license servers from any platform, and you do not need to install the RLM utilities - you only need a web browser.

Important Beginning in RLM v16.0, HTTPS is now used by default (HTTPS support was first added in v15.1). See enabling-https for more details.

In addition, the web server allows you to view ISV option files (with Manage permissions; from the action menu **?** of each ISV server), and the global RLM option file (as an Admin; from the Settings menu). Also, the web interface allows you to view the recent debug log information from any of the servers (via the action menu **?**) if you have Manage permissions. The **reread** and **shutdown** commands are possible with the Manage permission, while the View user can

see the status and usage for running ISV servers. See: access-control-web-interface for more on access permissions.

The web server is started automatically on port 5054 when RLM is started. To use the web server, simply point your browser to: https://ServerHostName:5054 and select the operation you would like to perform. You will be prompted for any required information.

Note For earlier version of RLM (15.2 and below) you may need to use http://ServerHostName:5054 to access the management interface.

If you would like to run the web server on a different port, specify the -ws NNNNN command-line argument when starting RLM, where NNNNN is the desired port. For example:

```
-ws 6000
```

The RLM web server is 100% self-contained in the RLM binary; no additional html files are required for operation.

For a description of the RLM web server UI, see the RLM License Administration Manual.

4.7.2 RLM startup options

The RLM command is:

```
% rlm [-c <license_file>] [-dat] [-dlog [+]<logfile>] [-info] [-l] [-noudp]
[-nows | -ws <port>] [-x [rlmdown|rlmremove]] [-v]
[-install_service] [-service_name <service_name>]
[-user <username> -password <password>] [-isv_startup_delay <seconds>]
[-verify] [-sslcert <certfile> -sslpriv <privkey>]
```

Table 4.5. These options can appear in any order on the command line.

Option	Definition
`-c <license_file>`	Specifies which license file to use. Overrides the setting of the RLM_LICENSE environment variable. The license_file parameter can be a directory containing license files, all of which will be processed.
`-dat`	Specifies that license files should have the extension ".dat", rather than ".lic". If -dat is specified, the RLM server will search for all files ending in ".dat" instead of ".lic" as documented elsewhere.
`-dlog <logfile>`	Specifies the pathname for the server debug log. Logfile will be overwritten unless preceeded by "+" character.
`-info`	Causes RLM to print information about all copies of RLM that are running on this computer, including copies which have run in the prior 24 hours, then exit.
`-l`	Causes RLM to only process command-line utilities from the local host.
`-noudp`	Causes RLM to not listen on its UDP port, so that client broadcasts do not work.
`-nows`	Instructs the RLM server to not start the embedded web server.
`ws <port>`	Instructs the RLM server to use `port` as the port number for the web server.
`-verify`	Causes RLM to start all ISV servers to verify their licenses, then all servers will exit.

-x [rlmdown | rlmremove]	Controls whether the rlmdown and/or rlmremove commands will be processed by the server. Specifying only -x will disable both commands.
-v	Causes RLM to print its version and exit.
-install_service	Installs RLM as a service in Windows. See Section 4.7.3.
-service_name	Sets Windows service name (default is "RLM").
-isv_startup_delay <seconds>	Speficies startup delay in seconds for Windows service. Useful if a license file specifies a hostid of type rlmid1 (hardware keys), the server is started at system boot time, and the key driver is not yet started at the time the ISV server needs to read it.
-user <username>	1. Windows expects the username argument to be <domain>\<user>. To use the local system domain, specify ".\<username>".
-password <password>	2. In order to run a service, the account specified by the -user argument must have the "Log on as a Service" property set.
-sslcert <certfile>	Location of certfile for using web server on HTTPS. Also requires -sslprivkey
-sslprivkey <privkey>	Specifies location of private key for using web server on HTTPS.

Note For more information on HTTPS, see Using SSL Certificates (HTTPS) with RLM.

Warning If the RLM server cannot bind the web server port (5054 by default), it will exit.

Warning If there is not at least one license file with the current hostname (as returned by gethostname()), or "localhost", a warning will be generated.

Report Log File

If you want to generate a report log file, specify this on an ISV-by-ISV basis in the individual ISV's options file. See the description of the REPORTLOG line in The ISV Options File for more information.

4.7.3 Running the RLM server as a service on Windows

On Windows servers, you may want to install and run the RLM server as a Windows service process. A service process can start automatically at boot time and remain running as long as the system is up, regardless of user logins and logouts.

You can install RLM as a service only in a command window. Once installed as a service, it remains installed until it is explicitly deleted as a service. Installing RLM as a service does not start RLM; services are started via the Windows Services control panel, and at boot time.

To install RLM as a service in a command window, use the RLM program itself (in a command window), with special arguments:

```
rlm -install_service -dlog [+]logfile [-service_name sname] <rlm runtime
args>
```

where:

- logfile is the pathname for the server debug log. **This parameter is required**. If

preceded by the '+' character, the logfile will be appended, rather than created.

- `sname` is an optional name for the installed service. If not specified, sname defaults to "rlm". If sname contains embedded whitespace, it must be enclosed in double quotes.
- **<rlm runtime args>** are any other command line arguments to be passed to RLM when it is started.

Example:

```
rlm -install_service -service_name rlm-xyz -dlog c:\\logs\\server.log -c
c:\\licenses\\xyz.lic
```

This installs RLM as a service under the name "rlm-xyz". When started via the Services control panel or at boot time, RLM will be passed the "-c c:\licenses\xyz.lic" args, and it will write its debuglog information to the file c:\logs\server.log.

Installed RLM services are also deleted with the RLM program. Services must be stopped via the command line or service control panel before they can be deleted.

Tip Deleting a service deletes it from the Windows service database; it does not delete the RLM executable or associated license file(s).

To delete:

```
rlm -delete_service [-service_name sname]
```

where:

- `sname` is an optional name for the installed service. If not specified, service_name defaults to "rlm". If service_name contains embedded whitespace, it must be enclosed in double quotes.

Notes:

- It is desirable to use the -c <license file> command line argument with RLM when installed as a service. Use of environment variables with Windows services is undesirable, as the environment passed to started services is the one in effect at boot time.
- On systems which run multiple RLM license servers, it is a good idea to install each ISV's instance of RLM with a service_name argument which reflects the ISV or ISVs whose licenses are being served by that instance of RLM. For example, if a system ran two instances of RLM as services, where the first instance served license for ISVs "Blue" and "Green", and the second instance served license for ISV "Yellow", they might be installed as "RLM Blue-Green" and "RLM Yellow", respectively.
- Because the Service Controller on Windows invokes services under a special user account in a special default directory, it is necessary to use full paths:
    - for the -c <license file> argument on the RLM command line
    - in ISV daemon paths in the license file
    - in options file paths in the license file
    - in debug log paths in the ISV options file
    - in report log paths in the ISV options file
    - for the -dlog debug_log argument on the command line
- When running as service, RLM changes its working directory to the directory where rlm.exe is installed. This is so that log files will be written there instead of in c:\windows\system32 as in prior versions (if log file paths are not specified as absolute paths.) rlm.exe checks to make sure that it can write to that directory before changing its working

directory. If it can't be written, RLM leaves its working directory as c:\windows\system32.

Important When running RLM as a service, it is strongly recommended that you specify a debug log location for each ISV server. This is done in the ISV Options File for each ISV server, using the DEBUGLOG keyword. If no location is specified for the debug log, the ISV server's debug information is lost when running as a service.

4.7.4 Starting the RLM server at system boot time on Unix systems

On most Unix systems, system services are started at boot time, usually via startup scripts located in /etc/rc.<something>. For example, on Solaris, the startup script might be placed in /etc/rc2.d/S98rlm. On Linux systems, the script could be located in /etc/init.d/rlm, with a link to /etc/rc5.d/S98rlm. Note that you must install this startup script as root. The startup script should su to a different user so that the RLM servers are not running as root. The following is an example of a script which would start RLM at boot time on Unix systems. Modify the first 5 variables for the target system.

Example startup script

```
#!/bin/sh
#
# rlm Start/Stop rlm
#

#-------------------------------------------------------------------
#-------------------------------------------------------------------
#-------------------------------------------------------------------
# NOTE:
# NOTE: Configure these 5 variables for your system
# NOTE:

# Set rlmuser to the user under which rlm will run
rlmuser=bobm

# Set rlmdir to the directory where the rlm binary is found
rlmdir=/home/bobm/rlm/dev/rlm

# Set rlmdir to the directory where the rlmdown binary is found
rlmdowndir=$rlmdir

# Set licfile to the path to the license file
licfile=$rlmdir/x.lic

# Set debuglog to the path to the debug log
debuglog=+$rlmdir/rlm.dl
#-------------------------------------------------------------------
#-------------------------------------------------------------------
#-------------------------------------------------------------------

start() {
echo $debuglog
                su - $rlmuser -c "$rlmdir/rlm -c $licfile -dlog
$debuglog &"
}

stop() {
                su - $rlmuser -c "$rlmdowndir/rlmdown RLM -q"
}
```

```
case "$1" in
        start)
                start
                ;;
        stop)
                stop
                ;;
        restart)
                stop
                sleep 2
                start
                ;;
        *)
                echo $"Usage: $0 {start|stop|restart}"
                exit 1
esac

exit 0
```

4.7.5 License Server Startup Processing

License servers use The License Environment to find their license file. In addition, any file whose name ends in **.lic** in the current directory when the RLM server is started (or when the rlmreread command is issued) is implicitly added to the end of the license file path. Finally, any file whose name ends in .**lic** in the directory where the *RLM* binary resides is added to the list of license files processed.

Note License files in the isv server's binary directory are not processed, only the RLM binary directory is searched.

License servers ignore *port@host* specifications in the License Environment. Once the list of license files is created, the license servers process all the resulting license files. The *RLM* server starts all ISV servers in all license files, and the ISV servers process and support all licenses in all license files with valid hostids.

When the RLM server starts, it uses the port # from the first file with the hostname on which it is running. The RLM server will attempt to use all the port #s in all the license files. It **must** be able to bind the port # in the first file. Once this is done, it attempts to use the port number from each additional file, logging either success or failure in the debug log. This means that when you receive a new product or license file, it can be installed in the application and RLM server directories without changing the port number in that file, which simplifies license administration.

ISV servers process all licenses in all files that have a valid *hostid* (by this we mean a *hostid* that corresponds to the computer on which the license server is running). The ISV servers attempt to combine licenses whenever possible - see the next section - and when combined the license counts add to create a single *pool* of licenses. ISV servers log (in the debug log) licenses with invalid signatures and licenses that will expire within 14 days. ISV servers do not process single-use (count==single) licenses.

ISV servers will detect that they are running on a virtual machine, and by default will refuse to run. The decision to run or not on a virtual machine is an ISV-by-ISV decision and is configured in rlm_isv_config.c.

If your server is configured to refuse to run on virtual machines, you can enable it for a particular machine by issuing an rlm_server_enable_vm license. This license can be any license that is valid on the server host (i.e., it can be either a nodelocked license or a floating license for that server), *however it CANNOT be locked to an Alternate Server Hostid*. So, for example, the following license would enable a license server (where the license is valid) to run on a virtual machine

through the end of 2023:

```
LICENSE ISVNAME rlm_server_enable_vm 1.0 31-dec-2023 1 sig=xxx
```

Note The rlm_server_enable_vm license will not be visible in status requests, or in rlm_products() calls.

4.7.6 Building Your ISV Server

Your ISV license server is built from components supplied by Reprise Software. You need to provide 2 custom inputs to the build of your license server:

- Your Public Key, for license key verification - rlm_pubkey.c – (see Section 4.3.4).
- A file of customizations for the server called rlm_isv_config.c (this file is contained in the src directory on the kit).

Once you have created these 2 files you create your ISV server by typing "make" in the kit directory.

Note You cannot create or use a server settings file if you use ISV-defined hostids.

There are 2 variables you need to set in **isv_config.c**:

1. Your **ISV name** ("demo" for demonstration purposes)
2. **use_flexlm_lockfile**

Only set "use_flexlm_lockfile" to a non-zero value if you want the RLM servers to lock out FLEXlm license servers and vice-versa. This would be done when you want to upgrade your customers' licenses and ensure that both license servers don't run. Note that the behavior of server locking is slightly different on Unix and Windows systems. On Unix systems, if either the RLM or the FLEXlm server is started, the other server will not run. On Windows, however, if the FLEXlm server is started first, the RLM server will run, and shortly after it starts (one to two minutes), the FLEXlm server will exit. If the RLM server is started first on Windows, the FLEXlm server will not run. This is due to the fact that the FLEXlm servers changed their locking mechanism, and the new servers check the old locks for a conflict but do not create them.

In addition, you can modify the parameters to the **rlm_set_cfg_set_compat() call to alter the compatibility of your ISV settings file with older and newer versions of RLM. Set the 2nd parameter to a non-zero value if you wish your settings file to work with older versions of RLM than the one in which you created the settings file. Set the 3rd parameter to a non-zero value if you want your settings file to work with newer versions of RLM than the one you used to create the settings file. The default is to enable newer versions, but not older versions.

Settings File

If you ship the settings file, you only need to ship one version of the file for all platforms which you support. This means that you do not need to have a supported server platform in-house in order to provide license server support for it.

4.7.7 ISV Server Open File Limits

ISV servers on Unix platforms will attempt to increase their open file limit. If a server is able to increase its open file limit, a line similar to the following will appear in the debug log when

the server first starts up:

```
mm/dd hh:mm (isvname) File descriptor limit increased from 256 to 65536
```

If you do not wish the ISV server to unlimit it's open descriptor limit, set the RLM_NO_UNLIMIT environment variable in the process where you run the server:

```
% setenv RLM_NO_UNLIMIT anything
```

4.7.8 How Licenses are Pooled by the ISV Server

When the ISV server processes all its licenses in the license file, it combines as many as possible into single pools of licenses. In order for 2 licenses to be combined into a single license pool, the following license fields must match exactly:

- Product Name
- Product Version
- License Disable specification (first checked in v6.0)
- License Options
- License Sharing specification (share and max_share)
- License Timezone specification
- License Platform list
- License Activation key (if specified)
- Both licenses must be counted or uncounted
- License node-locked hostid
- Both licenses must be user-based, host-based, or personal (or neither)
- Neither license can be a named-user license
- License password
- License id (also, an id of 0 will pool with any prior license with non-zero id)

Once pooled, the following fields are processed as shown:

Field	Result
count	Both counts added together
exp-date	Earlier date is remembered
hold	Maximum of the 2 values
max_roam	Minimum of the 2 values
max_roam_count	Minimum of the 2 values
min_checkout	Maximum of the 2 values
min_timeout	Maximum of the 2 values
soft_limit	Both soft_limit values added together
contract	If original is empty, use new
customer	If original is empty, use new
issuer	If original is empty, use new
type	If original is empty, use new

For all other fields, the field in the original license (i.e., the first to appear in the license file) is used.

Note In the case of the **contract**, **customer**, **issuer**, and **type** fields, once a license with a nonempty field is processed, no subsequent licenses in the license file will override this value, i.e., the first non-blank value will be the value associated with the license pool.

Note Different named_user licenses are never combined into one license pool.

License ID

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 nonzero id will only pool with the same ID# (assuming all the other attributes make it eligible to pool).

4.7.9 License Server Administration

There are various administration commands that can be used to cause the license servers to reread their license files, to remove licenses from certain users, etc. For a description of these administration commands, see license-administration-tools. In addition, options can be specified for each ISV server in isv-options-file. You can restrict access to administration commands via rlm-options-file.

4.7.10 Changing your ISV server name

Occasionally, an ISV needs to change their RLM ISV name. When this happens, you can set up your ISV server to lock out servers with one or 2 of your old name(s). To do this, modify your rlm_isv_config.c file to include a call to rlm_isv_cfg_set_old_isv_name() and rlm_isv_cfg_old_isv_name2() as follows (assuming your old ISV names were "oldisv" and "oldisv2"):

```
rlm_isv_cfg_set_old_isv_name(handle, "oldisv");
rlm_isv_cfg_set_old_isv_name2(handle, "oldisv2");
```

Remember to **remove the #if 0/#endif from around these calls**. If you only had one old ISV name, only include the first call.

Note If you have 2 old ISV names, and these servers don't make the calls above, it is possible for both of those ISV servers to run at the same time. So, for example:

- ISV names "a" and "b" are the old names.
- ISV name "c" is your new ISV name.
- Servers "a" and "b" make neither of the calls above.
- Server "c" makes both calls to lock out "a" and "b".

THEN:

- → servers "a" and "b" can run at the same time.
- → if either "a" or "b" is running, then either "c" won't run, or "a" and "b" will stop.

In the ideal situation, "a" locks out "b" and "c", "b" locks out "a" and "c", and "c" locks out "a" and "b". However, as long as "c" locks out "a" and "b", then "c" won't run if either "a" or "b" is running.

4.8 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 server binaries
- Install license file
- Start license server (if you are using floating licenses)
- Set up environment for users to find the license server when running your application.

In addition, your users would also want to know:

- Where to find the license administration tools.
- How to set up an options file for the license server. This step can be accomplished by providing them with the RLM License Administration manual.

4.9 Reprise Software's Recommended Software Installation steps

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

Warning Prior to v15.1, Reprise strongly recommended that you never install a license server as a privileged user (root or administrator).

4.9.1 Overview:

On the client side, i.e., 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. For floating, this license file will be used only to locate the license server host.

If your customers are primarily running on single network segments, you do not need to install license files on the client nodes, since the client will broadcast to find the license server.

Note This will not work if the client and server are located on different sides of a router, however, for small installations, the broadcast will save you having to configure license files on all client nodes.

If you ship floating licenses, install the server binaries and license file on the server node. The server license file doesn't need to be modified.

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 node-locked licenses, and only the server hostname needs to be set on the client side for floating licenses).

4.9.2 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 ship a nodelocked license:

- If you are shipping uncounted or single node-locked licenses, put the actual licenses into the license file. Install in the default directory. You're done.

When you ship a floating license:

- If you are shipping floating licenses, use a single HOST line in the license file for the client side. Use the default RLM port (5053) - which means you do not need a port number in this license file, and fill in the hostname with the name of the server computer at installation time. This license file should look like this:

```
HOST server-hostname
```

Note You can skip this step if you ship exclusively to small customers running on single-segment networks).

- On the server node, place the rlm binary, your ISV settings file (or ISV server binary), and the license file in a directory. This license file should have the real, signed licenses. The server hostname in this file can be "localhost", meaning that it doesn't have to be edited by the license administrator. The server license file's first two lines should look like this:

```
HOST localhost hostid
ISV your-isvname
```

  By configuring the license file this way, it does not need to be edited by your customer. These lines tell RLM to:

  - Use the default port (5053)
  - Use the ISV server settings or binary from the same directory as the rlm binary

- Include all your signed LICENSE lines in this file as well.
- Start the rlm server from the directory in the step above

If your customer already has another RLM server running:

- Install your ISV settings or server binary and the license file in the same directory as the other product's copy of the server binaries and license files, and do a "reread" operation on the running rlm. That's it, **however**:
- If your version of RLM is newer than the installed version, update the installed version to your version, then shut down the running rlm and start the new one.

If you ship new, additional licenses to your customer:

- Put the new license file in the same directory as the old one. If they are nodelocked licenses, put them on the client system. If they are floating licenses, put them into the directory with your other licenses and do an rlmreread on the license server.

Special note for MAC systems

On Mac systems, the RLM temporary directory changed in v14.1 from /var/tmp to /Library/Application Support/Reprise. See appendix-j for more information. One implication is that while this directory was writable in older macOS versions, it is no longer writable (at least on Catalina, and possibly on some earlier versions). So you will need to add a step to your installation procedures on macOS, to create /Library/Application Support/Reprise and set it to mode 777. If you do not do this, and the directory doesn't exist or is unwritable, you will get the RLM_EH_NOTEMPDIR error from rlm_init(), and when rlm is started, it will log this message and exit:

```
% rlm
        The RLM temporary directory:

        /Library/Application Support/Reprise

        could not be created.


        Please create this directory with the following 2 commands:

                sudo mkdir "/Library/Application Support/Reprise"
        and
                sudo chmod 777 "/Library/Application Support/Reprise"
```

4.10 Pre-Release Checklist

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

- Review the RLM API calls you make in your application to be sure that you use product names that are suitable (we strongly recommend using the name of the product that is in general use), and that the version numbers are correct. If you intend for your customers to be able to use old licenses from your product, be sure that the version number in the *rlm_checkout()* call is appropriate.
- If we have provided you with special debug libraries, make sure you use the non-debug libraries from the standard kit for your release.
- Review the options you used to Build Your License Server.
- Ensure that you have included the RLM Server, your ISV Server, and the RLM License Administration Tools in your distribution kit. If you are not using an ISV-defined hostid , ship the settings file rather than the isv server binary. Using the settings file means that your server-side executables are completely generic, and your customers can upgrade RLM versions and get bug fixes via a download from Reprise, and you have no ISV server build-test-release cycle to go through.
- Review the Section 4.4 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.
    - For information on installing RLMid1 devices see Section 6.4.
- A good practice is to include a folder for licenses in your installed product folder tree. In this folder, if you ship floating licenses, you would create a license file with a single HOST

line similar to the following:

```
HOST myserver any 5053
```

Note This HOST line does not need a valid hostid, only a correct port# and hostname). At runtime, your application passes the path to this directory as the first argument to rlm_init(). Then any license you ever issue - an expiring demo license, a production nodelocked license, or a license file simply containing a HOST line as described above - 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.

Chapter 5

Advanced Topics

5.1 Upgrading to a New Version of RLM

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

- First, Download the new RLM kit from the Reprise website - see details in Section 4.2.
- Then, unpack the kit and install. See details in Section 4.2.
- Next, copy the following 3 files from your old kit:

Filename	
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 for this version.

- You will have received an email from Reprise Software with your activation key and instructions for creating your new license (**license_to_run.h**). You only need to do this if you are upgrading to a new major version of RLM (e.g., from v13.x to v14.x), otherwise your old license_to_run.h will work.
- Edit the following 2 files in the new kit:

Filename	
src/license_to_run.h	Modify this to install the new RLM license you activated from the Reprise Software website, or use your old one if you are on the same major RLM version.
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 libraries, ISV server, rlmsign binary and activation binaries are ready to use.

5.2 Using RLM 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
- rlm<*vvmm*>.so on Unix/Linux

<*vvmm*> indicates RLM version, e.g., the DLL for v15.1BL2 is rlm1512.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 with various languages.

5.2.1 Using RLM with Fortran

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

5.2.2 Using RLM with Java

RLM Java is a number of Java classes presenting an object interface to underlying native code. The actual work in RLM Java is done by platform-dependent code accessed via the Java Native Interface (JNI). As such, RLM Java requires that a non-Java RLM kit be installed and configured before RLM Java can be used. The functionality available to a Java program is the same as that available to a C program. The RLM Java API is completely documented in Javadoc. Open doc/index.html with a browser to access that documentation.

The platforms that contain support for RLM Java are as follows:

- **All Windows** (x86_w3, x86_w4, x64_w3, x64_w4)
- **Linux Intel** (x86_l2, x64_l1)
- **Solaris Sparc** (sun_s1, sun64_s1)
- **Solaris Intel** (x86_s1, x64_s1)
- **Mac/Intel** (x86_m1, x64_m1)

If you would like to see the Java interface in action, follow these steps, after installing RLM on one of the platforms listed above:

On Unix, Linux, Mac systems:

1. Download the java_unix.tar.gz kit from the website.
2. In the platform directory, run "make shared". This builds the shared object necessary for supporting RLM Java. The name of the shared object is librlm<ver>.<shared>, where:
   <ver> is the major and minor version numbers and the build level. **<shared>** is the file type of a shared library. This is "so" on all platforms except Mac where it is "jnilib".
   For example, in RLM version 11.1BL2 on a Linux system the name would be "librlm1112.so"; on Mac it would be "librlm1112.jnilib".
3. Download the java_unix kit - java_unix.tar.gz.- and place it in your RLM SDK directory (i.e., in the same directory as "src", "examples", etc.).
4. Perform these steps:

```
% gunzip java_unix.tar.gz
% tar xvf java_unix.tar
% cd java_inix
% ./INSTALL
```

INSTALL will ask you which platform to install if more than one of the supported platforms is present. Select the platform corresponding to the machine you are running on.

5. In another window, in the platform directory, start up the RLM License server:

```
% rlm &
```

6. In the java_unix window, set the environment variable that lets the Java VM find the native library:
On Mac:

```
[export | setenv | etc] DYLD_LIBRARY_PATH[=]`pwd`
```

All others:

```
[export | setenv | etc] LD_LIBRARY_PATH[=]`pwd`
```

7. Run the example Java program:

```
% make runclient
```

This program, whose source code is in the java_unix directory, checks out a license from the license server and keeps it checked out until you hit enter.

On Windows systems:

1. Start the RLM license server:

```
> cd <platform>
> rlm
```

2. In another window, go into the java_win directory and configure the makefile if necessary. The makefile as shipped refers to the x86_w3 sibling directory. If you are running on a different Windows platform, edit the makefile and change the definition of "NATIVE_PLATFORM" to match the platform you're on. For example, if using 64-bit VC2015, change it to read:
NATIVE_PLATFORM = x64_w3
3. Run the example Java program:

```
> nmake runclient
```

This program, whose source code is in the java_unix directory, checks out a license from the license server and keeps it checked out until you hit enter.

5.2.3 Using RLM 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, e.g., 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
```

5.2.4 Using RLM with Visual Basic (outside .NET)

For information on integrating RLM with Visual Basic 6 applications see here.

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 rlm*VVVV*.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.

Note *VVVV* signifies the RLM version, such as 1512 for 15.1BL2

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, then 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
```

5.2.5 Using RLM with .NET

Overview

RLM provides a solution for .NET developers who want to use RLM to license their applications. It consists of a simple Interop layer that defines the RLM 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 license server, utilities, and the actual RLM code packaged in a DLL.
- Build the VS project "Reprise" in the dotnet folder on the SDK.
- Add calls to the RLM methods to the .NET application to be licensed.

Building the RLM .NET package

If you have VS2013 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 .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:

1. Open **dotnet\RLMTest\RLMTest.sln** in Visual Studio 2013 or later. If you have an earlier version of Visual Studio, create a new project for RLMTest, as described above for Reprise.
2. Build the project, either Release or Debug or both, but in the same configuration(s) as you built the Reprise project.
3. Copy rlm*VVVV*.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 1512 for 15.1BL2)

4. 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.
5. Run the application. It opens a command window for its 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 rlm*VVVV*.dll cannot be found.

Integrating RLM .NET into your Application

The RLM Reference manual (src\RLM_Reference.pdf) 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 functions are not supported in RLM .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

5.3 Debugging Licensing Problems in the Field

In order to diagnose a licensing problem in the field, you will need some information from your customer. Primarily, you need to figure out whether there are any nodelocked, uncounted licenses (including roaming licenses) which are available to satisfy your checkout request, and, if not, whether the application can connect to a license server to obtain a license.

There are 2 tools to help you diagnose these problems in the field:

- **client-side diagnostics** which show the application environment and local licenses available for checkout
- **server-side diagnostics** which show which licenses are available on the license server.

Product debugging information is also available.

5.3.1 Client-side Diagnostics

Built into every RLM client is the ability to output environmental information about the application's use of RLM. 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 Except on Windows, if you simply set **RLM_DIAGNOSTICS** without a value, the output will be sent to standard out - which may not be what you want. On Windows, if you want diagnostics in the console window, set RLM_DIAGNOSTICS=con in that window.

The resulting output will give the following information:

- Time the program was run.
- The 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. This list will have any roaming licenses listed first if RLM_ROAM is set, otherwise roaming licenses will be at the end of the list. In other words, the list will be in the order which RLM attempts to check them

out. Each license file will also have an indication of what license server would be contacted if no local licenses can satisfy the request.

Note For client-cached licenses, only the highest-numbered version will appear, even if there are lower-version licenses available. And only licenses cached by the same RLM version will appear in this output.

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

                RLM_CONNECT_TIMEOUT=<not set>
                RLM Reference Manual Page 94 of
296RLM_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>
                HTTP_PROXY=<not set>
                HTTP_PROXY_CREDENTIALS=<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>

Nodelocked/roaming licenses which can be checked out
        Roaming Licenses:

                test v1.0 OK

        In license file a.lic
                        (no server)

                test1 v1.0 OK
                test2 v1.0 error:-5
                test2 v1.0 error:-3
        In license file b.lic
                        (server at: 2800@host2)

                <none>
```

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.

5.3.2 Server-side Diagnostics

Pressing the "Diagnostics" button in the web interface will cause RLM to obtain the diagnostic information from all ISV servers, and put this information into a file ready for your customer to attach to an email message to send to your support department.

This information contains:

- The time the diagnostics were run.
- The working directory of rlm and the ISV servers.
- Relevant environment variables.
- A list of RLM's idea of the hostids on the machine where the license server is running (including your ISV-defined hostids)
- The license files in use, in the order RLM will use them.
- The contents of all license files.
- RLM status and statistics

For each ISV server:

- The server status and statistics.
- The license pool descriptions for all license pools.
- The list of users for each license pool.
- The RLM debug log file.

And, for each ISV server:

- The last 20kb of ISV server debug log output (or 20 lines if output is going to stdout).
- The last 20kb of report log output.

Note For each license pool, the parameters of the pool are displayed followed by a list of users of that set of licenses. The userlist is displayed in this format:

user:host:pid:isv_def:rprod:ver:normal:res:out:hold

where:

- **user** - is the username of the person holding the license
- **host** - is the hostname where the license is in use
- **pid** - is the process ID of the process holding the license
- **isv_def** - is any ISV-defined data provided to RLM before checkout
- **rprod** - is the requested product name
- **ver** - is the requested version
- **normal** - is the number of "normal" (ie, non-reservation) licenses in use
- **res** - is the number of reservations in use

- **out** - is the checkout time
- **hold** - is the license hold time

An example of this information is shown here.

```
RLM Server Diagnostics at 10/20/2009 13:58

Environment:

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

                RLM_CONNECT_TIMEOUT=<not set>
                RLM_EXTENDED_ERROR_MESSAGES=1
                RLM_LICENSE=2700@Localhost
                RLM_NO_UNLIMIT=<not set>
                RLM_PATH_RANDOMIZE=<not set>
                RLM_PROJECT=<not set>
                RLM_QUEUE=<not set>
                RLM_ROAM=<not set>
                RLMSTAT=<not set>
                RLM_LICENSE=<not set>
                HTTP_PROXY=<not set>
                HTTP_PROXY_CREDENTIALS=<not set>

        RLM hostid list:

                1d8bbd06 ip=172.16.7.13

License files:

                a.lic

============================================================
LICENSE FILE: a.lic ---- contents
============================================================
HOST paradise ANY 2700
ISV reprise reprise.set reprise.opt
LICENSE reprise foo2 1.0 permanent uncounted hostid=ANY _ck=c91efcffc7

sig="60P0450NFBEQT82V5DJTY7VPJSFXW70963B791R22G3TEVDBUUS8FE2MNJKHTW0
        K9DNED2D330"
LICENSE reprise test 1.0 permanent 2 _ck=13d7fcfe93 sig="60P04539JPQH1
        UHD8S4JHU55J4HTAX9371F9WSG22HRTERSD0RY5DMDE424TMEGR7GQHDGX730"

============================================================
Status for "rlm" on paradise (port 2700)
RLM software version v8.0 (build:1)
RLM comm version v1.1
debug log file: _stdout_

rlm Statistics --- Since Start --- Since Midnight --- Recent ---
Start time 10/20 13:58:12 10/20 13:58:13 10/20 13:58:13
Messages: 0 ( 0/sec) 0 ( 0/sec) 0 ( 0/sec)
Connections: 0 ( 0/sec) 0 ( 0/sec) 0 ( 0/sec)

ISV Servers
        Name port Running Restarts
        reprise 51391 Yes 0
```

```
============================================================

ISV reprise status on paradise (port 51391)
reprise software version v8.0 (build:1)
reprise comm version v1.1
debug log file: _stdout_
report log file: /user/matt/rlm/rlm/reprise.rpt

        reprise Statistics --- Since Start --- Since Midnight ---
Recent ---
        Start time 10/20 13:58:14 10/20 13:58:14 10/20 13:58:14
        Messages: 2 ( 2/sec) 2 ( 2/sec) 2 ( 2/sec)
        Connections: 1 ( 1/sec) 1 ( 1/sec) 1 ( 1/sec)
        Checkouts: 0 ( 0/sec) 0 ( 0/sec) 0 ( 0/sec)
        Denials: 0 ( 0/sec) 0 ( 0/sec) 0 ( 0/sec)
        License removals: 0 ( 0/sec) 0 ( 0/sec) 0 ( 0/sec)

License pool status ----------
Userlist fmt: user:host:pid:isv_def:rprod:ver:normal:res:out:hold

Pool 1: foo2 v1.0 permanent 0+0 soft:0 inuse:0
        h:ANY timeout:0 share:None trans:0
Usage for pool 1
matt:paradise:16253:::1.0:1:0:10/20 14:08:None
matt:paradise:16254:::1.0:1:0:10/20 14:08:None

Pool 2: test v1.0 permanent 0+2 soft:2 inuse:0
        h: timeout:0 share:None trans:0
No Licenses in use for for pool 2

============================================================
RLM debug log file last 20 lines
============================================================
10/20 13:58 (rlm) RLM License Server Version 8.0BL1

               Copyright (C) 2006-2009, Reprise Software, Inc. All
rights reserved

10/20 13:58 (rlm) License server started on paradise
10/20 13:58 (rlm) Server architecture: x64_s1
10/20 13:58 (rlm) License files:
10/20 13:58 (rlm) a.lic
10/20 13:58 (rlm)
10/20 13:58 (rlm) Using options file rlm.opt
RLM Reference Manual Page 97 of 29610/20 13:58 (rlm) Web server
starting on port 5054
10/20 13:58 (rlm) Using TCP/IP port 2700
10/20 13:58 (rlm) Starting ISV servers:
10/20 13:58 (rlm) ... reprise on port 51391
============================================================
End rlm debug log
============================================================


============================================================
ISV reprise debug log file last 20 lines
============================================================
10/20 13:58 (reprise) RLM License Server Version 8.0BL1 for ISV
"reprise"
10/20 13:58 (reprise) Settings from RLM Version 8.0BL1 for ISV
"reprise"
10/20 13:58 (reprise) Server architecture: x64_s1

        Copyright (C) 2006-2009, Reprise Software, Inc. All rights
```

```
reserved.

        RLM contains software developed by the OpenSSL Project
        for use in the OpenSSL Toolkit (http//www.openssl.org)
        Copyright (c) 1998-2003 The OpenSSL Project. All rights
reserved.
        Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com) All
rights reserved.

10/20 13:58 (reprise) Using options file reprise.opt
10/20 13:58 (reprise) Report log started on
/user/matt/rlm/rlm/reprise.rpt
10/20 13:58 (reprise)
10/20 13:58 (reprise) Server started on paradise (hostid: ANY) for:
10/20 13:58 (reprise) foo2 test
10/20 13:58 (reprise)
10/20 13:58 (reprise) License files:
10/20 13:58 (reprise) a.lic
10/20 13:58 (reprise)
10/20 13:58 (reprise) File descriptor limit increased from 256 to 65536
============================================================
End reprise debug log
============================================================


============================================================
ISV reprise report log file contents (/user/matt/rlm/rlm/reprise.rpt)
(last 20kb only)
============================================================
2 1.0 1 0 0 0 "ANY" "" "" "" "" "" 0 0 0 0
PRODUCT test 1.0 2 0 2 2 "" "" "" "" "" "" 0 0 0 0
10/19/2009 21:20

RLM Report Log Format 2, version 8.0, authenticated
ISV: reprise, RLM version 8.0 BL1
Logfile format Copyright (C) 2006-2009 Reprise Software, Inc.
For documentation on this format, email info@reprisesoftware.com
You are encouraged to build tools to process this data.

START paradise 10/19/2009 21:23:13
LICENSE FILE a.lic
PRODUCT foo2 1.0 1 0 0 0 "ANY" "" "" "" "" "" 0 0 0 0
PRODUCT test 1.0 2 0 2 2 "" "" "" "" "" "" 0 0 0 0
10/19/2009 21:23

        .....


============================================================
END ISV reprise report log file contents
(/user/matt/rlm/rlm/reprise.rpt)
============================================================
```

5.3.3 Product Debugging Information

To enable your application to output diagnostic information about any or all product names, 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

Warning DO NOT SET RLM_DEBUG WHEN RUNNING THE LICENSE SERVER!

This information can be obtained from the new rlmdebug utility (part of rlmutil), or directly from your application. If the RLM_DEBUG environment variable 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 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
```

5.4 Metered Licenses

Metered licenses are licenses that count the number of times something happens, sometimes

called *consumptive licensing*. For example, you might want to license the number of pages printed in your application or the amount of time your application runs. If that is what you want, then metered licenses are for you. A metered license is like a postage meter for software – you fill the meter up, then as the application runs, it consumes the meter count. When the meter is exhausted, subsequent checkout requests fail with RLM_EL_METER_NOCOUNT.

It's important to note that you do not have to change your code to make use of metered licenses. You call rlm_checkout() in the normal way. When you create the license, you specify that it is a *metered* license rather than *node-locked* or *floating*, and RLM does the rest.

Note Metered license cannot roam. In fact, if a license specifies any of the following attributes, it cannot be metered: hold, min_checkout, client_cache, max_roam, max_roam_count, soft_limit, user_based, host_based..

Also note that meter counts are 32-bit integers, so the maximum value for a meter counter is (2^{31})-1.

In addition, a meter counter number is also a 32-bit integer, so the maximum counter number is (2^{31})-1, however, in practice it is best to keep your counter numbers in the range of small integers.

5.4.1 About RLM meters

RLM metering is managed by the license server, so every metered license requires a license server to operate. The RLM metering implementation is **insecure**, for two reasons: the meters themselves are stored in files on the server computer, and users with access to the RLM web interface can add count to the meters.

Note This is not true if you use RLM Cloud, since the cloud servers are under your control and your customer does not have access to the standard web interface to modify the counters.

This implementation *will* work for you if you use RLM Cloud, run your license servers on your network to serve licenses for your customers, or if you want to give your customers the ability to meter licenses and then report on usage via the report log for post-use billing. But you cannot use RLM's metering implementation to **enforce** meter counts if your customers run their own license servers.

The license server maintains a single *meter*, which can contain any number of *meter counters*. It is the individual *meter counters* which your software will consume. An RLM metered license can specify an amount to subtract from an arbitrary meter on checkout, then an additional amount periodically while the software continues to run.

5.4.2 How to Specify a Metered License

In order to create a *metered* license, the count keyword is set to "meter". In addition, there are 4 optional parameters you can set to control the metering of the license:

Parameter	Description
meter_counter	The meter counter # used to meter this license
meter_dec	The amount subtracted from the meter on the initial rlm_checkout()
meter_period	The amount of time before an additional amount is subtracted from the meter
meter_period_dec	The amount subtracted from the meter each meter_period minutes

---

Note Metered licenses are not specific to a SERVER's hostid; they will work anywhere that you

configure the meter. In other words, the SERVER's hostid does not factor into the signature of a metered license.

If a metered license is specified with no additional options, the defaults are as follows:

```
meter_counter=1
meter_dec=1
meter_period=0
meter_period_dec=0
```

meter_counter=n

The meter_counter keyword specifies which counter is used for this product. Choose a positive integer less than 2^{31}. Smaller numbers are better for the RLM user interface.

meter_dec=n

The *meter_dec* keyword specifies the amount to be subtracted from the specified meter_counter when the server processes the rlm_checkout() request.

Note The *count* parameter of rlm_checkout() will multiply all the meter decrement counts. If the meter counter has insufficient count, rlm_checkout() will return RLM_EL_METER_NOCOUNT. Additional checkouts of a shared license will not decrement the meter.

---

meter_period=n

The *meter_period* keyword specifies the number of minutes before the counter is decremented again. Note that this time is approximate. The first periodic decrement will happen up to one minute earlier than the time specified. After that, the decrements will be at very nearly *n*-minute intervals. If unspecified or specified as 0, there will be no periodic decrement of the counter.

Note For shared licenses, **only one of a group** of shared licenses will have the meter decremented.

---

meter_period_dec=n

The *meter_period_dec* keyword specifies the amount to be subtracted from the specified meter_counter approximately every *meter_period* minutes. This is approximate because the license server does this subtraction in its periodic processing, which happens very close to once per minute. **If the meter counter has insufficient count at the time of this decrement, the server will decrement the counter (causing it to go negative), then drop the license** (similar to a TIMEOUT or an rlmremove operation), and your program will receive an RLM_EL_METER_NOCOUNT status the next time you call rlm_license_stat() on the license handle. The check-in status in the report log will have a reason code of 10, indicating that the check-in happened as a result of insufficient count in the meter.

Note The *count* parameter of rlm_checkout() will multiply all the meter decrement counts.

Also note that for shared licenses, only one of a group of shared licenses will have the meter decremented.

5.4.3 Example metered licenses

This example creates a metered license which subtracts 1 from counter number 1 on initial checkout and no further decrements as long as the program is running:

```
LICENSE reprise test 1.0 permanent meter sig=xxxx
```

This example creates a metered license which subtracts 20 from counter number 7 on initial checkout and an additional 5 every 10 minutes as long as the program is running:

```
LICENSE reprise test 1.0 permanent meter sig=xxxx
meter_counter=7 meter_dec=20 meter_period=10
meter_period_dec=5
```

5.4.4 Installation of Metered licenses

When a metered license is installed and the license server is started, the server creates the meter counter if it does not exist. Once created, the counter has 0 count, so checkouts that require meter count will not work. In the RLM web interface (in the license server status section), there will be a separate section for metered licenses - separate from all other licenses.

In the **Metered License Status** section, the metering parameters for each license are displayed, near the middle, along with a text box and a button to add count to the meter. Fill in the desired increment to the count in the text area, then press the "Add to counter N" button to its right. This will bring up a confirmation page, after which the meter counter is incremented.

The **Add Count to Meter** section of this form will only appear if the "edit_meter" privilege is not disabled in the RLM web interface for this user.

5.4.5 Reporting

When a metered license is checked out, in addition to the normal checkout record, an additional record is logged for the meter decrement. In addition, a decrement record is logged for each periodic decrement. These records are described in the RLM License Administration Manual. The reportlog can be used for post-use billing of metered licenses. The ability for your customer to control the meter count means that they have control of costs during the billing cycle.

5.5 Token-Based Licenses

Token-Based licenses are licenses that are defined 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) with a license count (or *uncounted*). On the other hand, if this were a *token-based* license, the product **write** would appear in the license file without a count, but with a specification of one or more other products which are used to satisfy the request. When the license server encounters a request for a **token-based** license, it uses the other products specified in the license to satisfy the request, rather than the originally requested product. These other licenses are called the *primary* licenses.

There are two main uses for token-based licenses. The first use of token-based licenses allows a license administrator to mix-and-match different products as their needs change. If several products are all defined in terms of a single pool of *primary* licenses, the license administrator can control license usage as needs demand. This can be a benefit to both ISVs and their customers, since as new products are introduced, if they are licensed based on the same *primary* license, all customers instantly have access to the new products without having to go through a purchase-order cycle.

Another use of token-based licenses is to allow alternate licenses to satisfy a request for a prod-

uct.To use the familiar example, if product **write** checks out a *write* license, the addition of a *token-based* 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 *token-based* 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.

A token-based license differs from a normal license in a few significant ways:

- The count field contains one of the 3 token keywords (**token**, **token_bound**, or **token_unlocked**) rather than an **integer**, **uncounted**, or **single**.
- The license has a token spec: **token="<prod ver count> … <prodN verN countN>"**
- The only optional parameter on a token-based license which is used by RLM is the start date. All other optional parameters are ignored.
- License Administration option processing is different for *token-based* licenses. See below.
- There are a few restrictions on *token-based* licenses, especially for queuing. See below.

Warning The keyword **token_locked** was deprecated in v12.4 and replaced with token_bound. Licenses with token_locked keywords will continue to operate, but they will not be locked to the server's hostid).

5.5.1 Types of token-based Licenses

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

```
LICENSE reprise test 1.0 permanent token token="<dollars 2.0 5>" sig=xxxx

LICENSE reprise dollars 2.0 permanent 10 sig=xxxx
```

This license is called a *simple token-based* license. Any *token-based* license that maps a checkout of one product into a (single) primary license is a *simple token-based* license.

A *token-based* license can map one request into multiple checkouts, however. In this case, it is called a *compound token-based* license. Using our product **test** as an example again:

```
LICENSE reprise test 1.0 permanent token token="<dollars 2.0 5> <cents 3.4
53>" sig=xxxx
LICENSE reprise dollars 2.0 permanent 100 sig=xxxx
LICENSE reprise cents 3.4 permanent 1000 sig=xxxx
```

Now, a request for 1 license of **test** v1.0 would result in the license server checking out 5 v2.0 licenses of the product **dollars**, and 53 v3.4 licenses of the product **cents**. If **both** of these primary licenses are available, the checkout request for test succeeds, otherwise it fails. Note that when a compound tokenbased license is checked out, the rlm_license_xxx functions return information about the first license in the list only. In this example, rlm_license_xxx functions would return information about the dollars license.

5.5.2 Issues with roaming

When a license is roamed, only the name of the license that was requested in the checkout can be used on the roamed system. What this means, in practice, is that as long as you use the same name to attempt the checkout, the checkout will succeed. However, sometimes token-based

licenses are used to migrate a product license name.

For example, if v2.0 of your product checks out an "old_name" license, and v3.0 checks out a "new_name" license, and you have a token license definition to map the "new_name" checkout to "old_name", then a roamed license created by checking out "new_name" will not work for the product that checks out "old_name", even though the "new_name" checkout may have been satisfied by an "old_name" license. In other words, while this mapping works on the server side, it does not work for a roamed license.

5.5.3 The License Count Keywords

In a *token-based* license, the count keyword is one of:

- token
- token_bound
- token_unlocked

token and *token_unlocked* imply that the *token-based* license itself does not include the license server hostid in its license signature. This makes the license usable in **any** license file.

Note token and token_unlocked are 100% equivalent.

---

token_bound means that the *token-based* license includes the license server hostid in its signature, and is valid only in *this* license file.

5.5.4 The token= keyword

In a *token-based* license, the *token*= keyword specifies the *primary* licenses which are checked out in response to a request for the *token-based* license itself. Specify one or more licenses to be checked out. These licenses can also be *token-based* licenses themselves (in which case the *primary* license(s) will be the ultimate expansion of all *token-based* licenses). The format is:

```
token="<product1 ver1 count1>[ <product2 ver2 count2> .... <productN verN
countN>]"
```

The request for the one of the original licenses turns into checkouts of:

- **count1** of product1, ver1
- **count2** of product2, ver2
- …
- **countN** or productN, verN

5.5.5 Nesting token-based licenses

The definition of a token-based license can include other licenses which are token-based licenses themselves. For example:

```
LICENSE reprise test 1.0 permanent token sig=xxxx token="<t2 2.0 5>"
LICENSE reprise t2 2.0 permanent token sig=xxxx token="<dollars 2.0 5>"
```

In this example, a request for one test v1.0 license results in 25 dollar v2.0 licenses checked out.

Note The license server uses nesting of greater than 20 levels to detect token "loops", so any licenses nested this deeply will be rejected. Also note that nesting has no effect on whether a *token-based* license is *simple* or *compound* - this is determined solely by whether a single request maps into a single checkout or not.

5.5.6 Restrictions on token-based licenses

There are a few restrictions on token-based licenses:

- All *token-based* licenses are processed by the license server, so there can be no uncounted, node-locked *token-based* or primary licenses that operate without a license server. (However, a license server can serve a node-locked, uncounted *primary* license.)
- All individual checkouts for a *compound token-based* license are satisfied by a single license server. This means that if a license turns into checkouts of *primary* licenses a, b, and c, where only a and b are available on one server and only c is available on a second server, the request will fail.
- The mapping from a token-based license to its *primary* license(s) maps to a single *primary* license. This means that a single token-based license generates requests of a **single primary license pool** (for each of it's primary licenses). So, for example, if your token-based license turned into a request for *primary* license *prim*, the first pool of *prim* licenses would be used, and other pools of *prim* licenses could not be used to satisfy this token request.
- Queuing is not allowed for *compound token-based* licenses. This would lead to license server insanity.

Note Token licenses, once fully resolved, should NOT contain multiple instances of the same primary license name. If this is the case, the server can inadvertently overdraft that license while processing the token-based checkout.

The following licenses are examples of licenses which contain multiple instances of the same primary license:

```
LICENSE reprise t1 1.0 permanent token token="<x 1.0 1><x 2.0 1>" sig=...
```

or:

```
LICENSE reprise t1 1.0 permanent token token="<t2 1.0 1><x 2.0 1>" sig=...
LICENSE reprise t2 1.0 permanent token token="<x 2.0 1>" sig=...
```

The server will generate a warning about this condition similar to the following:

```
01/01 10:09 (reprise) *WARNING: token definition for t1 has multiple
01/01 10:09 (reprise) instances of primary license x
```

For the 2 examples above, if there is only a single license for "x", a checkout of "t1" will cause the server to checkout 2 copies of "x" and return an RLM_EL_OVERSOFT error to the application.

5.5.7 rlmremove and token-based licenses

The *token-based* license itself cannot be removed. If any of the *primary* licenses are removed, the server will remove all *primary* licenses, and the application will notice a loss of license on

the next check with the server resulting from an rlm_get_attr_health() call or from the heartbeats generated by rlm_auto_hb(). In this sense, the *token-based* license works just like a regular, non *token-based* license.

5.5.8 Report Log

When a *token-based* license is checked out, the name of the license requested (and its version) is logged in the checkout record for each resulting *primary* license checkout. If the check-out results from the dequeuing of a previously queued request, the requested product name/version will not appear (they appear in the queue record, but not the checkout record).

5.5.9 License Administration Options

The only options that apply to the token-based license itself are the include and exclude keywords:

- INCLUDE
- INCLUDEALL
- INCLUDEALL_ROAM
- EXCLUDE
- EXCLUDEALL
- EXCLUDEALL_ROAM

All other license administration options have no effect on the token-based license.

All options affect the primary licenses, however.

5.6 Alias Licenses

Alias licenses are licenses that define themselves in terms of another license. You can think of an alias license as the nodelocked version of a token license.

Alias licenses are processed by the client only; they are never processed by the server. Here are some differences between token-based and alias licenses:

	Token-based License	**Alias License**
Where processed	Always on the server. Token-based licenses are ignored by the client.	Processed only by the client. Must be in a local license file. The server ignores alias licenses.
# of target licenses	Any number	One only
Target license(s)	Must be served	Can be nodelocked or served.
Nesting	Up to 20 levels	None allowed

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) with a license count (or *uncounted*). 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 can be either another nodelocked license or a served license.

An alias license is similar in syntax to a token-based license; both differ 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.
- License Administration option processing is different for *alias* licenses. See Section 5.6.8.
- There are a few restrictions on alias licenses, especially for queuing. See Section 5.6.6.

5.6.1 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 uncounted hostid=abcdef01 sig=xxxx
```

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

5.6.2 Issues with roaming

Since an alias license is processed by the client, **it cannot roam**.

5.6.3 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.

5.6.4 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 licenses turns into a checkout of *count1* of *product1*, *ver1*

5.6.5 Nesting alias licenses

Unlike token-based licenses, which can be nested 20 levels deep, **alias licenses cannot be nested**.

5.6.6 Restrictions on alias licenses

There are fewer restrictions on *alias* license than on a *token-based* license:

- 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.
- The mapping from an alias license to its *primary* license will attempt to find the primary license wherever it can, either node-locked, or on any license server.
- You can queue for the primary license of an alias(ed) license.

5.6.7 Report Log

When an alias license is checked out, if the primary license comes from a license server, the name and version of the *primary* license is logged.

5.6.8 License Administration Options

All options affect the *primary* licenses.

5.6.9 Use cases for alias licenses

There are three main uses for alias licenses, the 2nd and 3rd are the same as for token-based 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*.

The second use of an alias license allows a license administrator to mix-and-match different products as their needs change. If several products are all defined in terms of a single pool of *primary* licenses, the license administrator can control license usage as needs demand. This can be a benefit to both ISVs and their customers, since as new products are introduced, if they are licensed based on the same *primary* license, all customers instantly have access to the new products without having to go through a purchaseorder cycle.

A third 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.

5.7 Post-Use Billing

Many ISVs and License Administrators prefer a licensing option which allows the software to be used in excess of a predetermined limit, and then invoiced after-the-fact for actual usage. This capability has been referred to as "Pay-Per-Use", or "Metered" or "Post-Use Billing".

RLM supports this license model via the authenticated report log.

Note All report logs are authenticated.

RLM will generate authentication data within the report log. When the report log is returned to the ISV, the utility **rlmverify** is run to verify the contents of the report log. Usage is:

```
rlmverify report_log_file
```

rlmverify reads the specified *report_log_file* and checks the authentication records, then indicates whether the report log is correct or whether data in various sections has been modified.

5.8 License Roaming

In v14.1 Temporary Licenses were introduced, solving much the same use case, but in a more straightforward way. See Section 5.9 for more information.

RLM has the ability to allow a floating license to roam to a system which will subsequently be disconnected from the network for a short period of time. The resulting license can be used for the number of days specified when the license was set to roam, and is checked back in automat-

icallyat the end of this time. In addition, the customer can return the roamed license back to the license pool early if this is desired. Roaming is generally done with a checkout of a single license. If you request multiple licenses in a single checkout call, see the Section 5.8.4 section below.

Note Once the license is roamed to the disconnected system, by default the license will behave on that system as if it were a nodelocked, uncounted license. On the server system, one floating license will be assigned to the roamed system. The behavior of the roamed license on the disconnected system can be the same as a "single" license. In order to make the roamed licenses "single" licenses, make the following call in rlm_isv_config.c:

```
rlm_isv_cfg_set_roam_single(handle, 1);
```

In order to roam a license, an application performs a normal checkout with the RLM_ROAM environment variable set to a positive number (this number is the number of days which the license will roam). If successful, this checkout will create a roamed license which is valid for the specified number of days.

Note If ISVNAME_ROAM is set (where ISVNAME is the ISV's name in uppercase), it will be used in place of RLM_ROAM. In the remainder of this document, we will use the name RLM_ROAM to signify either RLM_ROAM or ISVNAME_ROAM.

If a subsequent checkout is performed with RLM_ROAM set to a positive number, the roam expiration date will be extended if required. The expiration date of the roam will not be truncated if the new value indicates an expiration sooner than the old expiration.

If you set RLM_ROAM to 0, then the client will use a roamed license if it is available, but if not, it will use a floating license.

Note If you request both roaming and queueing, the queue request will be ignored. In other words, if both RLM_ROAM and RLM_QUEUE are set, RLM_QUEUE is ignored.

A roamed license can be returned to the license server by setting RLM_ROAM to -1. When the license is checked out with RLM_ROAM set to -1, a successful checkout removes the roam data on the client machine and informs the server that the license is no longer roaming.

Warning In order for this to work, the license on the license server must be the same as it was when the roamed license was originally checked out. If the license is updated on the license server, the roamed license cannot be returned early, and must time out normally.

This is an important point – roamed licenses are specific to a particular license. If the license is replaced on the server side, the roamed license cannot be returned early – the roam can only time out in this situation. Roamed licenses are also version-specific – both license version and RLM version, so if your software is upgraded and either uses a newer version of RLM or requests a different version, the existing roamed license will not work.

RLM has the notion of the "oldest compatible version/revision" for roam file data. As long as the version and revision are greater than the oldest compatible version/revision stored in the roam file itself, the roam data will work on applications compiled with different RLM versions.

5.8.1 ISV control over license roaming

As an ISV, you control whether licenses are able to roam, and how long they can be checked-out in the disconnected state.

License Roaming is only available to systems that are able to check out an *rlm_roam* license

issued by you. This license must be available at all times - when the initial product checkout is done as well as when operating in a disconnected fashion. In other words, you must issue an uncounted, node-locked *rlm_roam* license. This license can, of course, be node-locked to "ANY" hostid.

The characteristics of the *rlm_roam* license also restrict how roaming can be used. For example, if the *rlm_roam* license specifies *platforms=x86_w*, then only 32-bit Windows systems would be able to check out roamed licenses. If *rlm_roam* specifies *max_roam=7*, then roamed licenses could only be checked out for 7 days. You can apply the *max_roam* attribute to any product license as well, but *max_roam* on the *rlm_roam* license creates a maximum for all products, so you only need to specify it once if you want to set a global maximum. If you do this and wish to disable roaming on an individual license, set *max_roam* to-1 for that license. You can, of course, always set an individual license *max_roam* to a value lower than the value in the *rlm_roam* license.

In addition, you can specify the *max_roam_count* attribute to any license. *max_roam_count* specifies how many of that particular license can be roamed. So, for example, to disable roaming of a particular license, specify *max_roam_count=0* on that license.

Note The hold and min_checkout parameters of a license which is roaming will be ignored.

5.8.2 License roaming and servers locked to transient hostids

If a license server is locked to a transient hostid (usually a dongle), it is easy for the license administrator to move the server to another machine, which not only allows for extra licenses to be roamed, but also interferes with the returning of roamed licenses, if the server isn't running on the same machine as when the license was roamed. Roaming is disabled if the server is locked to a transient hostid, and the application will receive an RLM_EL_NOROAM_TRANSIENT error.

If you want to enable roaming on servers with transient hostids, modify rlm_isv_config.c and set the 2nd parameter to the rlm_isv_cfg_set_enable_roam_transient() to 1.

5.8.3 Example rlm_roam license

The following rlm_roam license would allow anyone to roam any of your licenses for up to 14 days. You can apply max_roam to any individual license to override this 14-day default:

```
LICENSE isvname rlm_roam 1.0 permanent uncounted hostid=any max_roam=14
sig="xxx"
```

This license should always be available to the application which will be roaming. If you want to enable roaming for all your customers, a good way to do this is to sign the rlm_roam license and pass the signed license as the 3rd parameter to rlm_init(). If you only want to enable roaming for certain customers, then the license must always be available to the client in a local license file.

Once issued, the contents of the *rlm_roam* license, as well as the restrictions on the other product licenses will be in effect, but your customer controls roaming operation from that point on.

If you have issued an *rlm_roam* license and wish to disable roaming for a particular product license, set the *max_roam* attribute on the product license to -1. If you want an individual license to roam for only 5 days, set the *max_roam* on that license to 5. All other licenses will be able to roam for 14 days.

For a description of how roaming works from a license administrator perspective, see use-roaming-licenses.

5.8.4 Special Considerations with Roaming Licenses

Roaming licenses are attached to an exact license pool. This means that the licenses, once roamed, are associated with the exact LICENSE line on the server from which they were checked out. If that license is subsequently upgraded or replaced, the server will no longer have access to the roaming information. The roamed license on the client node will continue to be valid until it expires, and the server will return to its (newly upgraded) full complement of floating licenses. This also means that the server's host name and port-number must remain unchanged during the time the license is roamed. If you change either the server name or port, the roamed license cannot be returned early.

You should note that this means that a license which is roaming at the time the licenses are upgraded on the server can no longer be returned early, and as a consequence, a new roaming license (of a potentially higher version) will not be available until the original license roam expires. This also means that during the remaining time of the license roam, there will be one additional floating license available for each unexpired roamed license of the old version.

Another consideration is that a roamed license is a unit. Generally, it is unwise to roam licenses when a checkout request specifies multiple licenses. However, if you do this (checkout multiple licenses and allow them to roam), note that **any** check-in will check all the roamed licenses back in. For example, if you do a checkout of 4 licenses and they roam, then a check-in of 2 licenses with RLM_ROAM set to -1 will check in **all 4** of the licenses which were checked out and roaming. In other words, the checkout you do to roam a license, **even if the count is greater than one** creates a **single** roamed license. When this license is checked in (either by setting RLM_ROAM to a negative value, or at the end of the roaming period), **the entire quantity** of licenses is checked back in. Put another way, a roamed license is indivisible, and it is not possible to return a subset of the license count in a roamed license.

Since you cannot have more than one roamed license of a given name on a single system, if you first request a roam of N licenses, then you make a 2nd request to roam more than N licenses, the 2nd request will be rejected with an RLM_EL_ALREADY_ROAMING status. Otherwise, the first roam file is destroyed and can never be returned.

RLM provides a mechanism to remove a local roaming license when the application cannot check the license back in early. In order to unconditionally return the local roamed license, set the RLM_ROAM environment variable to -100 before performing a checkout of the license. If set to -100, the checkout processing will still attempt to return the roamed license early, but if the server cannot process the request, the client will still remove the local roaming license information.

5.8.5 Special note on roaming licenses from a broadcast-discovered server

Reprise Software strongly recommends that you do not attempt to use the broadcast method to locate the server if a license is to be roamed. While this will often work, there are circumstances where RLM cannot re-locate the original server which supplied the licenses, and the roamed license cannot be returned early.

If the broadcast server is the only license server on the network, RLM will be able to return these licenses. However, if there are multiple license servers on the network, there is no guarantee that the application will find the correct server to return the licenses, even if you manually set the port and host of the server before attempting to return the license. If you know the port and host of the ISV SERVER which roamed the licenses, you can then:

- Make sure the ISV server is running on the same port# (set the port # on the ISV line, if necessary).

- Set RLM_LICENSE to this port and host (the ISV server, not rlm).
- Set RLM_ROAM to -1 and check out the license again.

5.8.6 The relationship between roamed licenses, the rlm_checkout() count parameter, and rlm_isv_cfg_set_roam_single(handle, 1)

In nearly all cases, licenses will be roamed by using a checkout count of 1. However, it is possible to roam a license with a checkout count = n > 1. Reprise Software does not recommend this practice, but if you do it, take the following into consideration:

- *n* licenses will be subtracted from the license server's license pool
- Once roamed, if a request is made for <= n licenses, the roamed license will satisfy this request. This is not a counted license in the normal sense, it is just that the roamed license keeps track of the count of licenses requested.
- Once roamed, if a request is made for > n licenses, this request will fail with an RLM_EL_ALREADY_ROAMING error.
- If rlm_isv_cfg_set_roam_single() is called, only a single process will be able to check out these licenses, however that process can check out up to "n" roaming licenses.
- If rlm_isv_cfg_set_roam_single() is **not** called, any number of processes on the system will be able to check out up to "n" roaming licenses.

5.8.7 Extending the Roamed License

If your customer's plans change and they would like to keep the license until after the roaming time has expired, but cannot reconnect the system to the network, they can extend the roam by having someone on the network extend the roam in the RLM web interface. Note that this only works before the original roam period has expired.

This capability must be enabled by you.

Call *rlm_isv_cfg_enable_roam_extend(handle, 1)* in your rlm_isv_config.c file, and *re-build your ISV server or settings file* after you have done so. If you do not do this, the "Extend Roam" column will not appear in the RLM web interface. Also note that if you enable this capability, the *max_roam* setting from your *rlm_roam* license will have no effect on the length of a roam extension. What this means is that the *max_roam* setting of the roamed license itself will be what limits the extension of the roam. By default, every license has a *max_roam* setting of 30 days, unless you override it. So be aware that if you enable the roam extend feature **and** you specify *max_roam*, you must do so on the product license, rather than on the *rlm_roam* license for *max_roam* to have any effect on roam extensions.

To extend the roam, someone on the network views the license status and finds the license that is roaming. This user must have the "extend_roam" privilege to see the last field. By selecting Status->License Usage for this license, they will see a screen listing all licenses currently in use. By entering the # of days to extend the roam in the text box on the right and pressing "Extend", the process of extending the roam will start. Click **YES** to confirm the extension; the roam will be extended on the server side, and the screen will display the *roam extension code*, similar to the following:

```
test:12-sep-2017:1234567890abcdef1234567890abcdef1234567890abcdef
```

Now, on the disconnected client, set the environment variable RLM_ROAM_EXTEND to

the roam extension code:

```
setenv RLM_ROAM_EXTEND
test:12-sep-2017:1234567890abcdef1234567890abcdef1234567890abcdef
```

and run the application which checks out the "test" product. Once the checkout happens, the roam period is extended.

If you do not record the *roam extension code* from the web interface, it is also recorded in the server's debug log as follows:

```
08/28 15:45 (reprise) ROAM EXTENDED: 5 days test v1.0 by matt@zippy
08/28 15:45 (reprise) Roam extended for product test by 5 days
08/28 15:45 (reprise) Roam extension code, place in RLM_ROAM_EXTEND
08/28 15:45 (reprise) and run client to check out license:
08/28 15:45 (reprise)
test:12-sep-2017:1234567890abcdef1234567890abcdef1234567890abcdef
```

Note If you extend the roam past a daylight-savings-time transition day, you may appear to get one hour more or less than you might expect (when viewing the RLM web interface), but the roam actually ends at midnight.

5.8.8 Tutorial on Roaming Licenses

The RLM kit contains an example program to help you understand how roaming licenses operate.

This example is called **roam_example**, and it is built during the normal kit installation. The example license file on the kit also contains an **rlm_roam** license, so everything is ready to go.

We suggest you follow this procedure to familiarize yourself with the operation of roaming licenses.

1. First, install the kit.
2. If you haven't done so yet, sign the example license file using the command:

   ```
   % rlmsign example.lic
   ```

3. Start the license server (in a separate window on Unix, or in a command window on windows, so that it is easy to see the debug log):

   ```
   % rlm
   ```

Now you are ready to run the **roam_example** program to investigate RLM roaming license behavior.

The first thing we are going to do is to set up a roaming license. To do this, run **roam_example**, and when it asks for the RLM_ROAM value, enter 1. **roam_example** will set the environment variable RLM_ROAM to 1, then contact the license server with a checkout request for the **test1** license. You will note that **roam_example** now says that it has acquired a FLOATING license. This is correct. When the initial checkout which sets up the roaming license is performed, the application checks out a normal floating license, and informs the license server that it wishes for this license to roam. When this is successful, the roaming license is installed on the system where the application is running. You will note a line similar to the following in the server debug log:

```
01/14 15:28 (demo) OUT: test1 v1.0 by username@hostname (ROAMING for 1 days)
```

Finally, enter a <CR> to exit **roam_example**. You will note now that the server does *not* log the check-in of the license. This is because the license is roaming. The check-in will be logged later when either (a) the roam time expires, or (b) the license is manually returned (we will manually return the license later in this tutorial).

Your system is now set up with a roaming license, valid for 1 day. To test this license, run **roam_example** again, and specify 1 for the RLM_ROAM value. This time, you will see no activity in the server logfile, and **roam_example** will indicate that it acquired a ROAMING license. Enter a <CR> to exit **roam_example**.

Next, run **roam_example** and enter 0 or <CR> for the RLM_ROAM value. What happens next depends on whether the license server is still running or not. If the license server is running, **roam_example** will check out a floating license from the server and you will see this reflected in the server logfile. If the server is not running, then **roam_example** will use the roaming license, since no floating license was available. Why does it operate this way? Well, since the RLM_ROAM environment variable wasn't set, RLM attempts to the checkout in the "normal" way, which means it contacts the license server and asks for a license. Only if all the normal checkout methods fail does it then attempt to use a roaming license if one exists. Note how this varies from the case above, where you set RLM_ROAM to 1 - in that earlier case, RLM attempts to use any roaming license *before* contacting the license server, and only if it cannot find a roaming license does it check out a license from the server (and as part of that process create a new roaming license on the system).

Early Check-in

Finally, what happens if you have set up a roaming license but want to return it *early*. RLM provides for this by setting the RLM_ROAM variable to a negative value. Make sure the license server is still running (start it if it is not) and run **roam_example** again. This time, specify -1 when asked for the RLM_ROAM value. You will check out a floating license, and the server will remove the currently existing roaming license for this computer. When **roam_example** does its check-in (actually, it simply exits), the server will log the check-in of the originally-checked-out roaming license.

At any step, you can use the RLM web interface to examine the state of the license server to see which licenses are checked out.

One thing to note about this tutorial is that it worked because we pre-installed an **rlm_roam** license in the **example.lic** license file. With this license present, both the client and the server were able to check out **rlm_roam**, which is a requirement for roaming to operate. In practice, if you wish for your customers to have the ability to use roaming licenses, you will need to ensure that a valid **rlm_roam** license exists on any computer where a roaming license is created. Since the whole point of roaming is to allow disconnected operation, the disconnected node cannot depend on getting its **rlm_roam** license from the license server, so you will need to put a small local license file on that machine. In practice, the easiest way to do this is to create the license file in the same directory as your product binaries, since this license file only enables roaming, and should not vary from customer to customer. In other words, if you wish to enable roaming for all your customers, treat the license file with the **rlm_roam** license as a normal part of your distribution kit.

We encourage you to browse the source of **roam_example** - it is in the **examples** directory on the kit, named **roam_example.c**.

5.9 Temporary Licenses

You have the ability to move a floating license from an RLM Cloud license server and create a temporary nodelocked license which operates without an internet connection. Temporary licenses are always nodelocked, either uncounted or single. Temporary licenses are preferred over roaming licenses if they solve your use case.

This temporary license is much like a roaming license, but with a few important distinctions:

- There is no need for a separate enabling license, as is the case with the rlm_roam license for roaming
- There are no environment variables involved, rather, there are 2 new API calls to create and revoke the temporary license.
- The temporary license can be automatically refreshed when the computer is connected to the internet, without any user (or software) interaction.
- The temporary license is a standard RLM license, locked to a special dynamically created rehostable hostid, rather than to an encrypted roam file. This is because, for roaming, you have no control over the license server's environment, but for temporary licenses, the server is under your control.

In order to create the temporary license, you call the rlm_create_temp_license() API call. To return it early (before expiration), call rlm_return_temp_license() If any temporary license expires, the next call to rlm_create_temp_license() removes them. These calls are described in the Temporary license functions section.

5.9.1 Characteristics of a temporary license

As noted above, the temporary license is a normal RLM license, nodelocked to a dynamically created rehostable hostid. This license utilizes the RLM keywords:

- **exptime=hh:mm**
- **auto_refresh=refresh** parameters (used by RLM only, not visible in the normal client API)

The temporary license is stored in the RLM temporary directory, along with many other RLM files. RLM searches this directory for licenses before it searches any "normal" license files.

A temporary license has the standard RLM expiration date, but in addition, it has an expiration time on that date as well. It also has the auto_refresh parameters which specify how to refresh the license.

Note The auto_refresh keyword is reserved for RLM use only, and this data will not appear in any normal client-side calls such as rlm_products(), or rlm_license_xxx(). This keyword is not, and will not, be described in the license file section above.

Finally, temporary licenses cannot be created from a token-based license, only from regular base licenses.

5.9.2 ISV control over temporary license creation

Unlike roaming, no external license is required to enable temporary licenses, since you make an explicit call to set up or return the temporary license.

Also, unlike roaming, there is no issue of servers locked to transient hostids, since the server is always an RLM Cloud server. However, you cannot create a temporary license from a failover server, only from a primary server. If you attempt to create a temporary license from a failover server, you will get the RLM_EH_NOTEMP_FAILOVER error.

You should never create more than one temporary license for a product on a system – only one will be usable. In particular, 2 "single" licenses will not allow 2 separate checkouts of the license, so there is no point in creating more than 1 temporary license on a system.

If you attempt to create a temporary license from a checked-out license that did not come from an RLM Cloud server, you will receive the RLM_EH_TEMPFROMCLOUD error - "Temporary

licenses come from RLM Cloud servers only" (-187). This will happen, for example, if you check out an already-existing temporary license and then call rlm_create_temp_license() on the checked-out license handle.

5.9.3 If you want to extend the period of the temporary license

You can do this either manually, by calling rlm_remove_temp_license() then rlm_create_temp_license() or let RLM do it automatically for you by setting the auto refresh parameters. If you set auto refresh, then a checkout of the license within the number of minutes before expiration will attempt to refresh the license for the number of minutes to new expiration (from the current time – not an extension of the original expiration).

5.9.4 If you want to return the temporary license early

You do this by calling rlm_checkout() followed by rlm_return_temp_license(). As long as the license is available for checkout, and the software can still access the original server, the return will succeed.

Some additional notes

- You cannot create a license which expires later than the license on the license server itself.
- The license created will inherit several of the attributes from the floating license on the server:
    - version
    - options
    - contract
    - customer
    - issuer
    - akey

  Any other options in the original license are not preserved in the temporary license.
- Unlike a roamed license, the license server will attempt to associate this license with any license of the same name, version, and license options string. This means you can replace the license on your RLM Cloud server, and the temporary license will continue to be "checked out" on the server and returnable, *as long as the product, version, and options strings are the same*. If no license on the server has the same name, version, and options string, the automatic refresh of the license will no longer be successful. You will be able to return it, however this will only remove the license locally, since the server no longer knew about this temporary license.

5.9.5 Tutorial on Temporary Licenses

The code you write to create and revoke temporary licenses is more straightforward than roaming. To create a temporary license, you will check out a license, specify several parameters of the temporary license, and then call rlm_create_temp_license().

Check out the license as you would do normally. The license handle is passed to *rlm_create_temp_license()* in the RLM_TEMP_HANDLE. In the following example, we create a temporary "single" license valid for 3 days, which will be refreshed on checkout any time after 2 days before expiration. Note that the new license will expire 3 days from the time of refresh, not 3 additional days from the first expiration.

```
#define HOUR (60)
#define DAY (HOUR * 24)
char *prod = "product_name";
char *ver = "1.0";
RLM_HANDLE rh; /* You called rlm_init() earlier */

int
create_temp(RLM_HANDLE rh, char *prod, char *ver)
{
        RLM_LICENSE lic;

        RLM_HANDLE rh;

        int status = 0;

                rh = rlm_init(....);

                if (!rh) return(ERROR);

                lic = rlm_checkout(rh, prod, ver, ...);

                if (!lic) return(ERROR2);

                if (rlm_license_single(lic) || rlm_license_uncounted(lic))

                        return(GOT_NODELOCKED_LICENSE_NO_TEMP_CREATED);

                temp_handle = rlm_temp_new_handle(rh);

                if (!temp_handle) return(ERROR3);

                rlm_temp_set_handle(temp_handle, RLM_TEMP_HANDLE_DURATION,
(void
                *) 3*DAY);

                rlm_temp_set_handle(temp_handle, RLM_TEMP_HANDLE_WINDOW,
                (void *) 2*DAY);

                rlm_temp_set_handle(temp_handle,
RLM_TEMP_HANDLE_NEW_DURATION,
                (void *) 3*DAY);

                rlm_temp_set_handle(temp_handle, RLM_TEMP_HANDLE_LICENSE,
(void *)
                lic);

                status = rlm_create_temp_license(rh, temp_handle);

                rlm_temp_destroy_handle(temp_handle);

                return(status);
}
```

This license can be returned any time it can be checked out. In other words, it cannot be returned after it expires, but this happens automatically, anyway. This example returns the license created in the earlier example:

```
lic = rlm_checkout(rh, prod, ver, ...);

status = rlm_remove_temp_license(rh, lic)
```

In this example, if the temporary license has expired and your checkout is granted a license from the server, the rlm_remove_temp_license() call will return RLM_EH_NOTTEMP (-76).

5.10 Personal Licenses

RLM's Personal Licenses are licenses that are locked to a particular username with an optional hostname. They are much like user-based or named-user licenses, however your customer is able to maintain the user list via the RLM Cloud portal's GUI. Actually, you have the option of allowing your customer to maintain the list, or you can maintain it, depending on whether you give your customer access to the portal or not.

RLM Personal Licenses are available using *RLM Cloud* license servers only.

In order to provision a particular customer, browse to the RLM Cloud portal (known as the "*RLM Cloud Control Customer Portal*"), then select the **Personal Licenses** tab, located under the **Admin** tab. Next, if there is more than one product using personal licenses, select the product. At this point, you will see a form that allows you to enter the customer's name/password/options. When complete, press **Save User List** at the bottom. This will save the *currently displayed page only*. You can also cancel if you have made a mistake, or re-select the product using the buttons at the bottom of the form.

There will be as many "user slots" as there are licenses for this product.

5.10.1 How to use Personal Licenses in your application

If you want to use personal licenses you do 2 things:

1. Create a license with the keyword "personal=*NNN*", where *NNN* is the # of personal licenses. *NNN* should match the count on the LICENSE line.
2. In your application, gather the user's username and password, using whatever method is appropriate. Force the username (including the hostname) to lowercase. Set the username and hostname using the rlm_set_environ() call, and set the password using the environment variable RLM_PERS_PW, using a call similar to rlm_putenv("RLM_PERS_PW=your-user's-password");

So, for example, if your user's name is "testuser@testhost.com" and the password is "testpw", make the following 2 calls:

- rlm_set_environ(handle, "testuser", "testhost.com", "");
- rlm_putenv("RLM_PERS_PW=testpw");

Note Personal licenses and Dynamic reservations use the same variables for transport, so it is not possible to have a personal license that uses dynamic reservations. This is a minor restriction, since the use cases for these 2 kinds of licenses are so completely different.

5.11 Client-Side License Caching

RLM has the ability for a license to be cached on the client side, so that subsequent requests do

not require communications with the license server. You can think of this as a very short-term, automatic, roam.

To enable this functionality, you do two things:

1. Call a setup function in rlm_isv_config.c
2. Add the optional "client_cache=xxx" attribute to the license

You should note that only licenses which have a sharing attribute **without a share count** will be usable as cached licenses. In other words, if the license has a "share=u" attribute, and the same user attempts a checkout, the cached license can be used. However, if the "share=" attribute is missing, or the share attribute is something like "share=u:8", no checkouts of the cached license will be possible. Also note that the host attribute is matched automatically, by definition, since the license is being used on the same computer.

5.11.1 Enabling the functionality

To enable this capability, call:

```
rlm_isv_cfg_enable_client_cache(handle, 1);
```

in rlm_isv_config.c If you do not make this call, your application won't write the data required by subsequent runs to avoid communications with the license server.

There are 2 new errors: RLM_EH_CACHEREADERR and RLM_EH_CACHEWRITEERR if the local cache file cannot be read or written.

Using client_cache

Once the functionality is enabled, add the "client_cache" attribute to the licenses which you wish to have this capability. For example:

```
client_cache=600
```

will cause the license to be cached on the application's computer for 10 minutes. client_cache is specified in seconds, and the maximum time is 3600 (1 hour).

That is all you have to do. The RLM library implements client caching for you, and subsequent checkouts will use the cache file rather than attempting to contact the license server for a checkout.

rlm_license_xxx() and rlm_product_xxx() functions for use with client_cache

Function	Description
rlm_license_client_cache()	Returns the value of the client_cache parameter from the checked-out license.
rlm_license_cached()	Returns 1 if the checked-out license is a client_cached license, 0 otherwise.
rlm_product_client_cache()	Returns the value of the client_cache parameter from the license.

5.11.2 Server processing

Any license with a client_cache specification is treated by the license server **identically** to a license with a min_checkout specification. This is true whether or not you enable the client-side with rlm_isv_cfg_enable_client_cache(). The server does no other special processing for client_cache, only the minimum_checkout processing.

Note If you don't call rlm_isv_cfg_enable_client_cache() in rlm_isv_config.c, the server will still hold the license for the cache period, but the application won't write the cache file, so subsequent checkouts will use the license server.

Notes on License and RLM versions

Licenses of different versions can be cached, and licenses checked out by software linked with different RLM versions can be cached as well. Software linked with a particular RLM version will only see licenses that were cached by the same version. This means that if you have 2 programs, linked with different RLM versions and checking out the same license, you will see a separate checkout at the server for each program, then subsequent invocations of each program will use the corresponding cached license until expiration.

RLM_DIAGNOSTICS

Client Cached licenses will appear in RLM_DIAGNOSTICS output, however, only the highest version license enabled for any given product name will appear. And only a license which was cached by a program using the same version of RLM.

rlm_products()

Client cached licenses **will never appear** in the list of licenses returned by rlm_products().

5.12 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:

- 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.
- You cannot use RLM Cloud servers.

Reprise Software recommends using the **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 ISV server settings file and avoid building an ISV server.
- You can use an Activation Pro generator settings file and avoid building a custom generator.

- You can use RLM Cloud to provide cloud-based license servers to your customers.

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.
- This hostid can't be used for the license server itself, only for node-locked licenses.

5.12.1 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.

Note This hostid can't be used for the license server itself.

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

5.13 Failover License Servers

RLM provides the capability for a license server to take over the license complement of another server which has gone down. The server whose licenses are being taken over is called the *primary server*. The server which takes over the license load of the *primary server* is called the *failover license server*. During the time that the failover server is serving the licenses, no roaming operations are permitted on the licenses.

Note When using failover servers, there cannot be a firewall between the two servers.

Note Failover servers are not supported on HP/UX or IBM AIX systems.

The ability for a server to take over the load of another server is selected by an ISV on a customer-bycustomer basis, and enabled by an *rlm_failover*, *rlm_failover_server* or *rlm_failover_server_activate* license in the license file of the server that is to take over.

The difference between an *rlm_failover* license and an *rlm_failover_server/rlm_failover_server_activate* license is whether RLM checks for the machine being down (*rlm_failover*) or the RLM process on the machine being down (*rlm_failover_server/rlm_failover_server_activate*) before serving the licenses from the primary server. In addition, the *rlm_failover_server_activate* license requires a license activation from an Activation Pro server before proceeding to serve the primary server's licenses. With this license, you can see in your ActPro database how often a particular customer's failover servers have been activated.

While the *failover license server* can be serving its own compliment of licenses, Reprise Software recommends that it should have no licenses of its own, but simply be standing by, waiting for one (or more) other server(s) to fail so that it can take over. In the remainder of this chapter, we will use the term **failover license** to mean either the *rlm_failover*, *rlm_failover_server*, or *rlm_failover_server_activate* license.

5.13.1 Enabling a Failover Server

In order to enable a failover license server, the ISV issues a **failover license** with the following characteristics:

- **count=***some non-zero value*
- **hostid=**<*hostid of the primary server*>
- **_primary_server=**<*hostname of the primary server*> (rlm_failover licenses)
  <**OR**>
  _primary_server=<*host or port@host of the primary server RLM process*> (rlm_failover_server or rlm_failover_server_activate licenses)
- **akey=***activation-key* – this is the activation key to be used to activate in the case of an rlm_-failover_server_activate license.
- **issuer=***url-of-activation-server* - this is the URL of the activation server to be used to activate in the case of an rlm_failover_server_activate license.

Note A failover_server/failover_server_activate license can specify just the hostname of the server, in that case, the default port (5053) is used.

5.13.2 Configuring Failover License Servers

Reprise Software recommends configuring *failover license servers* as stand-alone servers that do not serve their own complement of licenses. In other words, Reprise recommends configuring a license server that has **only failover licenses** for the other license servers on the network. In general, one license server configured in this way should be sufficient to support failover of all other license servers on the network.

The exception would be a case where each individual license server is serving thousands of clients, in which case we recommend that you configure a *failover license server* for each one or two of the normal license servers.

Note Failover license servers do not support license roaming operations. Actually, this is determined on a license pool by license pool basis - any license pool that contains any licenses from a primary server will not support any of the roaming operations.

Example Failover License Server license file

The following license file would specify a license server that is acting as a failover server (on node **failover_host**, hostid **11111111**) for the license server on hostid **12345678** (node **primary_server**):

```
HOST failover_host 11111111 5053
ISV reprise
LICENSE reprise rlm_failover 1.0 permanent 1 hostid=12345678
        _primary_server=primary_server sig="x"

## Alternate LICENSE line:

LICENSE reprise rlm_failover_server 1.0 permanent 1
        hostid=12345678 _primary_server=5053@primary_server sig="x"

## Alternate for an activated failover server:

LICENSE reprise rlm_failover_server_activate 1.0 permanent 1
```

```
        hostid=12345678 _primary_server=primary_server
        issuer=hostedactivation.com akey=1234-5678-9012-3456
        sig="x"
```

Note The hostid in the LICENSE line for the failover license is the hostid of the server on the node primary_server).

Note The "_primary_server" keyword was called "_failover_host" prior to RLM v9.2. "_primary_server" is an alias for "_failover_host", which will continue to work. However, RLM versions prior to v9.2 will only recognize the "_failover_host" keyword to specify the primary server name/port #.

The license file on "primary_server" should be a normal RLM license file, without the rlm_failover licenses. This license file MUST be present and readable by RLM on the failover machine as well, otherwise the failover server will not serve the licenses from the primary when it goes down. So, for this example, a license file on the primary server would look something like this:

```
HOST primary_server 12345678 5053
ISV reprise
LICENSE reprise product1 1.0 permanent 3 sig=....
LICENSE reprise product2 1.0 permanent 7 sig=....
 ...
LICENSE reprise productN 1.0 permanent 2 sig=....
```

5.13.3 Installing Failover License Servers

When you receive the failover license file, do the following to install your *failover license server*:

1. Install *RLM*, your *ISV server* and the *primary server license files* on the *primary server*. Start RLM.
2. Install *RLM* and your *ISV server* on the *failover license server* node.
3. Install all license files from the *primary server on the failover license server node*.
4. Edit the license file with the **failover license** to put the hostname of the *primary server* in the *_primary_server*= field, and the hostname of the *failover license server* on the HOST line.
5. Install any ISV options in an options file and make it accessible to the *ISV server* either through its license file or by giving it the default options filename of *isv.opt*.
6. Ensure that *RLM* will process all the license files above and start *RLM*.
7. Ensure that your license administrator's client software sees the failover license server's license file, i.e., put a license file with a HOST line for the failover server *where your application will find it*. If you omit this step 6, the failover license server will take over, but your application will not be able to check out a license from the failover server.

5.14 Special notes on rlm_failover_server_activate licenses

The *rlm_failover_server_activate* license requires a machine which can connect to the internet to operate. Before serving licenses, the failover server must activate its special license (an *rlm_failover_auth* license) before it will begin to serve licenses from the primary server's license file.

The server will log a warning at startup if there is an *rlm_failover_server_activate* license in its license file but it cannot verify that the activation key is good (the server calls rlm_act_key-valid() to validate the key).

Note When you create the activation key to be used with the rlm_failover_server_activate license, specify a product name of "rlm_failover_auth", and a nodelocked, uncounted license. Your ISV server will pass the failover server's hostid as the hostid in this activation request.

Note Reprise Software recommends setting _primary_server to **localhost** (or **5053@localhost**) when issuing the *failover* license. This value would be changed by the license administrator to the name of the *primary server* upon installation. Setting this to **localhost** ensures that the *failover license server* will not be activated by mistake if the license is not edited.

The **failover license** must be a **counted** license; otherwise, the server will not use it. (If it were not a counted license, the **failover license** could be used on any server, which could result in multiple servers taking over **at the same time** for the primary server.)

In order to enable the *failover license server*, the **failover license** needs to be in one of the license files it is using, and the license file(s) for the *primary server* also need to be processed by the *failover license server*.

5.14.1 When a license server encounters a failover license, it does several things:

- Starts a separate thread in the license server to periodically monitor the health of the *primary server*:
    - If the license is an *rlm_failover* license, the failover server attempts to determine whether the license server *machine* is running, by connecting to ports on that machine. If the license server machine is up but the license server is not running, the failover license server will **not** take over serving licenses.
    - If the license is an *rlm_failover_server* license, the failover server attempts to connect to the main RLM port on the primary server. The main RLM port is specified in _primary_server on the rlm_failover_server license, or defaulted to port 5053 if not specified. If the failover machine cannot connect to the RLM process on the primary server, the failover server takes over.
- If the primary server should go down, enables all licenses in license files for the *primary server* by performing the equivalent of an *rlmreread* command. In the case of an *rlm_failover_server_activate* license, the failover server will attempt to activate its license from the Activation Pro server before proceeding. If successful, it will serve the primary server's licenses, otherwise it will not.
- When the *primary server* comes back up, disables all licenses in license files for the *primary server* by performing the equivalent of an rlmremread command.

Note Failover License Servers are by no means secure and are intended to be used by trusted customers. If the _primary_server value is set to a non-reachable system name/server, the Failover License Server will always serve the licenses from the primary server. The rlm_-failover_server_activate license at least informs you when this is happening via the fulfillment logs in ActPro.

5.15 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.

5.15.1 Windows

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

5.15.2 Linux/Solaris

Create a shared object (.so) that contains the code for your product as well as the RLM 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 -versionscript=file option.

On Solaris, replace "-Wl,–version-script=*file*" with "-D *file*"

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 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 objects will remain globals in your .so file.

5.15.3 MAC

On Mac, you create a dynamic library, and use the "exported_symbols_list" linker option to list the global symbols you want to export. From the macOS ld man page:

-exported_symbols_list filename

> The specified filename contains a list of global symbol names that will remain as global symbols in the output file. All other global symbols will be treated as if they were marked as __private_extern__ (aka visibility=hidden) and will not be global in the output file. The symbol names listed in filename must be one per line. Leading and trailing white space are not part of the symbol name. Lines starting with # are ignored, as are lines with only white space. Some wildcards (similar to shell file matching) are supported. The * matches zero or more characters. The ? matches one character. [abc] matches one character which must be an 'a', 'b', or 'c'. [a-z] matches any single lowercase letter from 'a' to 'z'.

5.16 Internet Activation

5.16.1 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.

In the case of a node-locked product, 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.

Floating licenses would operate in a similar manner, except that the number of floating licenses to be activated is required.

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.

5.17 Server-Server License Transfer

RLM has the ability to transfer a set of licenses from one license server to another. This capability is described in the transferring-licenses section of the License Administration manual. Licenses are said to be transferred from a source license server to a destination license server.

The **destination** license server is *always* an RLM server. As shipped from Reprise Software, the **source** license server is an RLM license server as well. However, as an ISV, there are two things you can do to enhance the License Transfer capabilities of RLM:

1. License transfers can be "disconnected", which means that the destination license server acquires roaming licenses from the source license server.
2. You have the ability to extend the license transfer capability to enable transfer from a non-RLM **source** license server. This capability is useful, for example, when you are doing a transition from another license manager to RLM, to avoid the possibility of issuing your customers duplicate licenses.

Note You can define a single non-RLM license transfer extension, i.e., you can only support one other license manager for license transfers.

Support for these 2 scenarios are described in the next sections.

5.17.1 Disconnected license transfer support

Disconnected license transfer, allows your customer to move some licenses from the source server to a destination server for a period of time, and the destination server does not need to maintain contact with the source server during that time. Internally, disconnected license transfer uses RLM's *license roaming* capability, so all the controls and restrictions of license roaming apply:

- You must issue an *rlm_roam* license to your customer to enable this capability (although we provide a means for you to do this internally in your server so that no separate license has to be shipped).
- Once a license is transferred to the destination server in this way, the license on the source server cannot be removed or upgraded or the destination license will not be able to be returned to the source server (until it automatically expires).
- Your control over the maximum roam duration influences your customers maximum time to disconnect the destination server's licenses.
- Your customer's control over ROAM_MAX_COUNT and ROAM_MAX_DAYS in the ISV server options file will limit the number of licenses the destination server can request, as well as the duration of the disconnection.
- These licenses, like all transferred licenses, are not eligible to be roamed to another system.

The description of disconnected license transfer under transferring-licenses.

If you already issue *rlm_roam* licenses to your customers, then you need to do nothing further – your customer will only have to put the *rlm_roam* license in a license file which the destination server reads at startup time, then they will be able to create license transfer definitions with "Days to hold license" (in the license transfer GUI) set to a non-zero value.

Should you wish to make the delivery of the *rlm_roam* license even easier, you can add the following call to your rlm_isv_config.c file, by providing a valid, signed rlm_roam license:

```
rlm_isv_cfg_set_server_roam(handle, "<LICENSE isvname rlm_roam 1.0 uncounted
hostid=any
sig=xxxxxxx>");
```

Note This license must use the following parameters: * version: "1.0" * exp: "permanent" * count: "uncounted" * hostid: "any" * NO other parameters

If you use this call and you ship an ISV server settings file, you will need to re-generate your settings file with RLM v10.1 or later.

Note These licenses, once roamed to the destination license server, are not available for checkout by an application on that server machine which doesn't contact the license server – they are only available for checkout by contacting the destination license server itself.

5.17.2 License Transfer from a non-RLM source license server

Should you wish to support a source server that is not an RLM license server in order to do a transition of your customers, you would do the following:

- Integrate RLM into your application, just as any new RLM customer would.
- Ship your RLM-licensed product with new RLM licenses to all new customers, and to all existing customers who purchase additional licenses. This is exactly what an RLM customer who did not have a previous license manager would do.
- Ship your RLM-licensed product, **without new licenses** to existing customers. Using license transfer, the customer can partition their existing license pool to use some licenses with new products (via the license transfer capability) and some old licenses with old products which utilize your old license manager.
- Eventually, as the old versions of your product are no longer used, the need for license transfer becomes obsolete when all your customers are using new products with new (RLM) licenses.

Implementing ISV-defined license transfer

In order to enable license transfer from a different **source** license management system, you need to write the code to perform initialization, checkout, check-in, status, heartbeats, etc., from the other license management system. RLM provides a plugin interface for you to enable this in your ISV server. We call this **ISV-defined license transfer**.

In order to implement **ISV-defined license transfer**, there are 2 steps you need to take:

1. Write the functions to do the transfer, and to register with the RLM ISV server.
2. Link your license transfer functions into your ISV server.

There is an example of **ISV-defined license transfer** code on the kit in the **examples** directory. The code is contained in the file **rlm_transfer.c**. This module illustrates **ISV-defined license transfer** by implementing license transfer from a source RLM license server. While this code is not useful (since RLM transfer is built into RLM), it illustrates how to write the transfer functions in terms of the RLM API.

Writing your ISV-defined license transfer code

You begin by writing a source module to perform the other license manager's basic checkout/check-in operations. Reprise Software recommends using the rlm_transfer.c example code as a guide to writing this code. This code has been integrated and tested with RLM by Reprise

Software, and is tested on each release as part of our normal QA test suite.

In this module, you will write 9 independent functions for your license manager "xxx":

- check_auth_xxx()
- fillin_auth_xxx()
- update_status_xxx()
- open_xxx()
- checkout_xxx()
- hb_xxx()
- checkin_xxx(),
- close_xxx(), and
- rlm_ix_enable()

These 9 functions are described in the sections which follow.

Reprise recommends that you put all source into a single source file, and make all the functions static/private with the exception of rlm_ix_enable(), which must be a public function. You must define all 9 functions.

Once you write this source module, include the object in the makefile to link your ISV server. So, for example, if you name your module "my_xfer.c", and the other license manager's libraries are named "other_lm.lib", modify the makefiles as follows:

Change the lines:

```
ISV_XFER_SRC =
ISV_XFER_OBJ =
ISV_XFER_LIBS =
```

to:

```
ISV_XFER_SRC = my_xfer.c
ISV_XFER_OBJ = my_xfer.obj
ISV_XFER_LIBS = other_lm.lib
```

In addition, you may need a special rule to build "my_xfer.obj" from "my_xfer.c". Create this rule here as well.

Parameters for all transfer functions:

Type	Name	Description
(void *)	auth	RLM license auth handle (used only to pass through to other RLM functions).
int	count	# licenses transferred
(void *)	license_handle	The license manager's license handle.
(void *)	lm_handle	The license manager's (job) handle.
(void *)	ls	RLM license server handle (used only to pass through to other RLM functions).
(char *)	product	Product name transferred.
(char *)	server	Server where transfer originated.
int	status	Checked-out license status on update.
(char *)	ver	Version of product to transfer.

check_auth_xxx() - check parameters of transferred license

```
int check_auth_xxx(void *ls, char *product, void *lm_handle, void
*license_handle)
```

This function verifies that a checked-out license is valid for a transfer. Several things could make it invalid - e.g., if it is a user- or host- based license, we don't want to transfer it. Returns 0 if the license is OK, or -1 if it is invalid.

For RLM licenses, we make sure the license is not a user-based, host-based, named-user, or token-based license.

For other license managers, other types of licenses may be ineligible for transfer. You can decide what is, and what is not transferred by accepting or rejecting it here.

- product: product name
- lm_handle: lm handle (RLM_HANDLE for rlm, cast to void *)
- license_handle: license handle (RLM_LICENSE for rlm, cast to void *)

If you return -1, your check-in and close function will be called by RLM (checkin_xxx()).

fillin_auth_xxx() - fill in RLM license parameters for transferred license

```
void fillin_auth_xxx(void *ls, void *lm_handle, void *license_handle, void
*auth, void *product,
int count)
```

This function fills in a license authorization from the checked-out license. License parameters are contained in "lm_handle", and/or "license_handle".

Some other license managers (e.g. FLEXlm) don't have a license handle but use the lm handle along with the product name to identify the checked-out license.

update_status_xxx() - update the status of a transferred license

```
int update_status_xxx(void *ls, char *server, int status, void *lm_handle,
int *passes_to_check)
```

This function updates the transfer status if a checkout fails.

Returns 0 if the checkout can be attempted again, or 1 if it will never succeed.

Also, passes_to_check should be updated with the number of passes before another checkout should be attempted. Each pass is one minute later. So, for example, if the license server is down, we wait 10 passes (10 minutes) before attempting the checkout again in the standard RLM license transfer code.

open_xxx() - open the other license manager

```
int open_xxx(void *ls, char *lmname, char *host, void **lm_handle, int
*unavailable)
```

This function creates the license manager's handle that will be used for the license checkout, and sets any required attributes in that handle.

Returns the handle in the "lm_handle" parameter. So, in the example RLM code, this returns an RLM_HANDLE in the "lm_handle" parameter.

checkout_xxx() - check out the product from the other license manager

```
int checkout_xxx(void *ls, void *lm_handle, char *product, char *ver, int
count, void **license_handle)
```

This function performs the license checkout of product/ver/count. The function return is the license checkout status, and the license handle (if applicable) is returned in "license_handle". If the license manager has no license handle (e.g. FLEXlm), return NULL for the license_handle.

hb_xxx() - perform a heartbeat on the other license manager

```
int hb_xxx(void *lm_handle, void *license_handle)
```

This function performs a heartbeat on the handle; returns status:

- 0 for good status
- LM error for error

Some license managers (e.g., RLM) require the license handle for the heartbeat, others (e.g., FLEXlm) require the license manager's (job) handle. Choose the handle appropriate for your license manager.

checkin_xxx() - check in the product and close the license handle

```
int checkin_xxx(void *lm_handle, void *license_handle)
```

This function performs the product check-in and closes and frees the license handle.

The license handle should be closed and freed if the license management system requires this. In RLM, for example, an rlm_checkin() call does the check-in and frees the license handle.

close_xxx() - Close the license manager's handle

```
int close_xxx(void *lm_handle)
```

This function closes and frees the license manager's handle.

The license manager's handle should be closed and freed if the license management system requires this. In RLM, for example, an rlm_close() call performs this function.

rlm_ix_enable() - enable ISV-defined license transfer in RLM

This function registers your ISV-defined license transfer functions.

The function name, "rlm_ix_enable" should not be changed, nor should the names of the 2nd and 3rd parameters to the call.

Call rlm_ix_enable as follows:

```
rlm_ix_enable( "license manager name",  /* License Manager name */
        rlm_ix_transfer,                /* DON'T CHANGE THIS */
        rlm_ix_xfer_done,               /* DON'T CHANGE THIS */
        open_xxx,                       /* Your handle open fcn */
        checkout_xxx,                   /* Your checkout fcn */
        fillin_auth_xxx,                /* Your auth fillin fcn */
        check_auth_xxx,                 /* Your license check fcn */
        update_status_xxx,              /* Your status update fcn */
        hb_xxx,                         /* Your heartbeat fcn */
        close_xxx,                      /* Your checkin fcn */
        checkin_xxx);                   /* Your LM handle close fcn */
```

RLM support functions

There are 2 support functions you will need when writing your ISV-defined license transfer code. These functions are used in the example, and are described here:

```
rlm_ix_log(void *ls, char *errtxt)
```

logs a message to the ISV server debug log. Pass "ls", the RLM license server handle, from the calling parameters, and a string to log.

```
rlm_ix_update_auth(void *ls, void *auth, int what, int ival, char *sval)
```

updates the server's license "auth" data with parameters from the transferred license.

"what" is which parameter to update (see license.h for definitions), if the value of "what" is an integer, put the value into ival, if it is a string, put the pointer into sval. See rlm_transfer.c in the examples directory for examples on use of rlm_ix_update_auth().

5.18 Virtualization

RLM provides capabilities to enable and/or restrict the usage of both applications and license servers in virtual environments.

By default, RLM allows **applications** to run in virtual environments. Also, by default, RLM does not allow **license servers** to run in virtual environments.

You can restrict application usage in virtual environments by using the "disable=vm" keyword in the license. See Disable Computing Environments for more information.

RLM's default behavior regarding running servers on VMs is designed to prevent accidental/uninformed/unintended use of them. It's not that we actively discourage the practice - it's more that we want you to think about your policy regarding VMs so you make an informed decision. The problem of course is that it is trivial to clone a VM image, hostid and all, and run multiple copies of it in an enterprise, thus gaining multiple sets of licenses if a license server runs there.

Control over whether license servers run in a virtual environment is in two places - you set the default in rlm_isv_config.c with the *rlm_isv_cfg_set_enable_vm()* call. If the second parameter is 0 (the default), then your servers will not operate in virtual environments. If it is set to any non-zero value, your servers will all run on virtual machines.

If you want to enable license servers for only certain trusted customers, you can leave the default in rlm_isv_config.c set to disable servers on VMs, and issue an **rlm_server_enable_vm** license for the individual machine for which you want to enable. So, for example, the following license would enable a license server (where the license is valid) to run on a virtual machine through the end of 2025:

```
LICENSE ISVNAME rlm_server_enable_vm 1.0 31-dec-2025 1 sig=xxx
```

Note The rlm_server_enable_vm license will not be visible in status requests, or in rlm_products() calls.

Note RLM treats Docker containers on non-Windows systems the same as virtual machines.

5.19 Cloud Computing

If you would like to serve your licenses in the cloud, Reprise Software recommends using our *RLM Cloud* service offering. Everything is set up for you to manage your licenses on servers provisioned by Reprise with *RLM Cloud*, without having to provision servers and configure the server farm yourself. While RLM can do this, the setup is tricky, which is why we created *RLM Cloud* in the first place.

5.19.1 What is RLM Cloud?

RLM Cloud is a hosted license server farm, managed by Reprise Software. With *RLM Cloud*, you provision software license servers and deploy licenses to your customers which do not require any installation by your customer's administrators. Since you deploy *both the license servers and*

the licenses in *RLM Cloud*, there is no need for Activation Software such as RLM's *Activation Pro* if you are using *RLM Cloud*.

When you use *RLM Cloud* to run your license servers, there is never an issue of customers wanting to move a license server or rehost an application. All the licenses are under your control in *RLM Cloud*. Issues related to license servers running on virtual machines, dongles to lock license servers, spoofed hostids, and other issues related to license servers being under the control of your customers all go away with *RLM Cloud*.

5.19.2 How RLM Cloud Works

In *RLM Cloud*, you manage a set of **license server hosts** which run the **license server processes** for your customers. The management software is called *RLM Cloud*, and it is a web-based application, hosted on Reprise servers. *RLM Cloud* is a pure Software-as-a-Service offering - there is no software for you or your customers to install.

One thing to note is that there is nothing special about the license server processes that run in *RLM Cloud* – they are standard RLM servers. What this means is that your applications, built using RLM, will operate equally well with the traditional on-premises license servers or with *RLM Cloud*. This also means that you can deploy a mixture of on-premises and *RLM Cloud* servers, as your customer requirements dictate. You should note that there are some special management interfaces in the copy of the rlm server that runs *RLM Cloud*, but your ISV server is identical.

5.19.3 Requirements for using RLM Cloud

While *RLM Cloud* uses the standard RLM servers, support for *RLM Cloud* in the license file is required. This means that your application must use RLM v12.0BL2 or later, and RLM v12.1BL2 or later is recommended.

Please contact your Reprise Software Salesperson for more information.

5.20 Disconnected Operation

Sometimes it is necessary to support many thousands of clients connected to the license server. If your customers generally will have more than 5,000-10,000 client applications using licenses at the same time, you might want to use RLM's *disconnected operation* capabilities. Reprise does not recommend using *disconnected operation* if your customers generally have a relatively small number of clients in use at any one time.

To utilize RLM *disconnected operation*, call the alternate initialization routine *rlm_init_disconn()* in place of *rlm_init()*. *rlm_init_disconn()* takes 4 parameters – the first 3 are the same as *rlm_init()*, and the fourth is a promise of how often you will contact the server by calling *rlm_get_attr_health()*. The *rlm_init_disconn()* call is described in the rlm_init(), rlm_init_disconn() section.

RLM *disconnected operation* works much the same as permanent connections up to the point of the first checkout. Once the license is checked out, however, RLM will disconnect the client from the license server. Each subsequent call to rlm_checkout(), rlm_checkin() or rlm_get_attr_health() will re-establish the connection to the server, send the appropriate message, read the response, then disconnect again.

There are several considerations to keep in mind if you use disconnected operation:

- Heartbeats are slower. In normal operation, the first heartbeat is sent immediately after the first checkout, then each call to *rlm_get_attr_health()* reads the heartbeat response,

which should be waiting to be read, then sends the next heartbeat. In this way, the client never has to wait for a round trip to the server. With *disconnected operation*, however, the client must establish a new TCP/IP connection to the server, send a message, and read the reply.

- A handle set to *disconnected operation* cannot be later set to use permanent connections. If you require a permanent connection to the license server, call *rlm_init()* and create a new handle.
- Since the client is disconnected, the RLM server cannot send reverse heartbeats to the client, and does not receive notification if the client exits from TCP/IP. This means that the license will be held on the server side for the *promise* interval from the last time the client contacted the server, even if the client has exited. The license server will also time out the disconnected client and clean up their data structures if it has not heard from the client within the *promise* interval.
- Since *disconnected operation* is used by ISVs with large numbers of connected clients, the minimum interval at which you can contact the server is larger. Whereas for connected handles, the minimum heartbeat interval is 30 seconds, for *disconnected operation*, it is 10 minutes. This also means that the **minimum promise interval** is 10 minutes.

Note Reprise Software recommends setting promise to at least 20 minutes to avoid unintended license removal by the license server.

- Pay particular attention to the relationship between the minimum heartbeat interval and your *promise*: rlm_get_attr_health() can be called as often as you like, but it will not communicate with the server if it was called less than 10 minutes earlier. So, for example, if you have a *promise* of 15 minutes, and you call *rlm_get_attr_health* every 9 minutes, you will not talk to the server for 18 minutes (since the first call will not communicate, and the 2nd will happen 18 minutes after the last call that did communicate). In this example, you will have exceeded your *promise*, and the server *may* time out your application and forget about you. (We say "may" because it depends on when the server actually does the timeout processing, which is not precise, since it only does this processing every 5 minutes).
- The license server processes timed-out clients every 5 minutes. This is not synchronous with your checkout request, however, it is every 5 minutes from when the server started.
- If you plan to check any licenses in then close the handle (i.e., if you are not going to use the handle after checking a license in), then you should omit the *rlm_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 by only calling *rlm_close()* RLM will only reconnect to the server one time for all your licenses as well as to tell the server that you're are done with the handle.

5.20.1 Notes on *Disconnected Operation* for RLM

We performed some tests on an Intel Solaris server. Our results indicate the following:

- In our testing (non-shared licenses on a 2007-vintage server machine), the license server is capable of processing ~100 transactions/second with a load of 10k clients.
- At 25k clients, the server can process 50 transactions/second, and at 50k clients, 35-40 transactions/second.
- To first approximation, a transaction can be either a checkout or a heartbeat.

What does this mean to you when designing your product checkout/heartbeat strategy?

Let's say you have a product with an average usage time of 20 minutes (1200 seconds). If you want your server to support 50k clients, this implies a (steady-state) checkout rate of 42 checkouts/second. From the above, you can see that this is as many transactions as the server can process, leaving none for heartbeats, so this is not a realistic expectation. Also, if there is a large burst at certain times of the day, the server might well get so overloaded that client requests time out.

A more practical limit might be 25k clients or less. At 25k clients, the sustained checkout rate is 21/second. If you set the heartbeat interval to 20 minutes as well, the heartbeat rate would be 21/second (actually, it would be less, since some clients would exit before ever generating a heartbeat), making the total transaction rate at the server 42/second, which would leave the server with some headroom for slight bursts of activity. If you tried to send heartbeats every 10 minutes, however, the server could do little more than process heartbeats (at 42/second) from the 25k clients. Again, you would have to consider checkout rates at peak demand times of the day.

5.21 How RLM Clients Find the License Server

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

- **A HOST (or SERVER) line in a local license file. specified by:**
  - 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.
- If none of these methods work to locate a license server, the RLM client will broadcast to find a license server on the local area network.

Note 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()*.

If you use a license server, this license file need only contain a single HOST (or SERVER) line with the correct hostname (and port if you don't use the default port). The hostid in this license file does not matter. So, for example, if your license server is on machine *paradise* at the default port (5053), this license file can contain this single line:

```
HOST paradise any
```

5.21.1 Broadcasting to find the license server

RLM clients can broadcast on the local network to find a license server.

Note Broadcasts do not travel through routers, so this is only effective for smaller networks.

This can be quite useful for installations at smaller customer sites which have no full-time IT personnel.

The broadcast has the following characteristics:

- If a license server for your ISV name is located using any other method, then the broadcast will not be performed.
- The first server to answer the broadcast will be used. Generally, since this is a small network, there will only be one.
- The broadcast will always happen on UDP port 5053, which has been assigned to Reprise Software by the Internet Assigned Numbers Authority (iana.org).
- The client waits for a reply for 2 seconds. This is not configurable.
- You can disable this broadcast in your software by calling rlm_isv_cfg_disable_broadcast() in your rlm_isv_config.c file (or any time after you call rlm_init() and before you check out a license).

5.22 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 in wide-character environments on those systems. On Windows however, where the operating system stores path information in widecharacters, RLM must be wide-character-aware.

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. RLM stores them internally as UTF-8 and converts back to wide characters before using them in filesystem operations. In this way, RLM clients and servers can be installed on wide character paths and work correctly.

Note RLM does not support wide characters in the ISV options file or in debuglog and reportlog file names.

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.

5.23 Alternate Server Hostids

The *Alternate Server Hostid* capability works in conjunction with RLM Activation Pro and is not supported when used without ActPro.

5.23.1 Overview

An Alternate Server Hostid is a method to lock a license server to a nodelocked, uncounted license, rather than directly to a hostid. The main uses for this are to lock the license server to a rehostable hostid (so your customer can revoke and re-authorize the server), or so that you can disable a license server that is in the field.

The *alternate server hostid* hostid type (license=serial#) is supported by RLM only on

the HOST/SERVER line. A single ISV server can only use a single Alternate Server Hostid locked to a rehostable hostid, i.e., there cannot be two different "license=xyz" hostids in 2 different license files if the two Alternate Server Hostids both use rehostable hostids. Also note that you cannot use an Alternate Server Hostid for an *rlm_server_enable_vm license* (either as it's nodelocked ID or as a floating license where the server uses an ASH hostid).

The license server, at startup time, will verify the license specified by the *alternate server hostid*, and if it is valid, will then perform the operation specified by the "check-type" parameter. The "check-type" parameter specifies whether automatic communications with the activation server are done or not, and if so, how often the activation server is checked. If "check-type" is "none", no automatic checking is done. If "check-type" is "startup", the license server will make an rlm_act_keyvalid() call to the activation server at license server startup time. If "check-type" is "daily", the license server will check at startup and again at midnight on every subsequent night. Finally, if "check-type" is "6hour", the server checks the license on startup and again every 6 hours. Once the license is verified and any checks are done successfully, the server starts up normally. If a midnight (or 6 hour) check fails, the license server will continue to run for "tolerance-1" checks. If no successful check is done within "tolerance" days/6 hours, the server will exit.

Setting "check-type" to "startup", "daily", or "6hour" allows you to turn off a license server by disabling the associated activation key. If the check fails with a "key not valid" error the server will exit immediately (on startup), otherwise, the tolerance parameter specifies on how many days (or 6-hour checks) errors can occur. On the failure the "tolerance" time, the license is disabled.

The license for the *alternate server hostid* must have the following characteristics.

Note This is the alternate server hostid license only, these restrictions do not apply to the other licenses you put in the license file.

Also note that this license is generated correctly by Activation Pro when you select the "Alternate Server Hostid" license type, so you don't have to worry about these parameters (other than setting the product name correctly), which are here for reference:

- It must be a nodelocked, uncounted license (ActPro does this automatically when you select the "alternate server hostid" license type).
- Product Name: rlm_server (you must set the product name correctly)
- options string, as defined below
- serial# must be a decimal number of <= 15 digits (activation pro generates this automatically every time it creates an activation key)
- issuer=URL-of-activation-server (if you use either the startup, daily, or 6hour option, see below)
- hostid: required
- akey= required (if using "startup", "daily", or "6hour")

The options string has the following format:

```
options=serial#:check-type:tolerance
```

Where:

serial# - is the serial number from the license=serial# specification on the SERVER line

check-type is one of the 4 values:

- none: no check with the activation server is done by the license server
- startup: check (rlm_act_keyvalid()) with the activation server at license server startup time
- daily: check (rlm_act_keyvalid()) with the activation server on startup and

every night at midnight.

- 6hour: check (rlm_act_keyvalid()) with the activation server on startup and every 6 hours subsequent to that.
- tolerance: is a # of days (or number of 6-hour checks), N. Used only for the "daily" and "6hour" check-types. The server can fail its check N-1 times in a row without any issues, N failures in a row cause the server to stop serving licenses for this hostid.

Note The serial number is the part of the hostid that makes the floating licenses in the license file unique, so if you create an alternate server hostid license without Activation Pro (which is not recommended or supported) you should BE SURE to give each of your customers a different serial number. Activation Pro will do this for you automatically.

When creating the product definition for an Alternate Server Hostid license, use ActPro's "Alternate Server Hostid" License type in the product definition for this – this product definition will generate the SERVER, ISV, and nodelocked LICENSE lines required. Activation Pro also allows you to enter the check type, and tolerance in the product definition. The serial number is generated automatically when an activation key is created, and cannot be changed. You MUST, however, set the product name to "rlm_server".

When creating your product definition, be sure to select "Yes" in the "Include Activation Key in License" parameter, and if you are using this capability to provide rehostable licenses, select only "rehostable" in the "Allowed Hostid Types" selection of either the product definition or activation key.

If you use the *alternate server hostid* on the SERVER line, a single instance of rlm can only support your ActPro server, it cannot support other ISV's servers.

Finally, when you create an alternate server hostid activation key, you almost certainly want to give it an activation count of 1. If you give it a count > 1, your user can create multiple servers which have the same hostid. This means that any license you issue to this hostid will work on each of the servers so activated.

In all use cases for alternate server hostids, the plan is to activate the alternate server hostid first. Once this is activated, call rlm_act_fulfill_info() on the activation key to retrieve the server's hostid that was generated (it will be of the form "license=xxx"). You can then use this hostid when activating your product licenses.

> **Warning** If you use alternate server hostids, you MUST set the URL in the ActPro database in the admin->database tab.

5.23.2 Server startup tolerance window

In addition to the tolerance count for 6-hour and daily checks, RLM v14.1 allows you to specify a time window from the last successful contact with the activation server. If you specify this window, and the license server is unable to contact the activation server at startup time, the license server will still start up.

Note The daily and 6hour checks will still be performed and if the license server remains unable to contact the activation server for the number of times specified, the license server will stop serving licenses.

To set the server startup tolerance, call rlm_isv_cfg_set_ash_start_tolerance(), with the # of minutes you wish to allow the tolerance interval to have. A good value would be 1 or 2 days. For example:

```
#define DAY 60*24

        rlm_isv_cfg_set_ash_start_tolerance(handle, DAY*2);
```

5.23.3 Alternate Server Hostid Examples

Example 1. Lock your server to a regular hostid, and check the license every day.

In this example, the server will check the activation key with the activation server every time it starts, plus every night at midnight. If the check fails for 10 days in a row the server will stop serving other licenses for this hostid. You would use a SERVER line and license like this:

```
HOST localhost license=1234
LICENSE isvname rlm_server 1.0 1-jan-2025 uncounted
hostid=12345678 issuer=http://www.hostedactivation.com
akey=1234-5678-9999-4321
options=1234:daily:10 signature
```

Example 2. Lock your server to a rehostable hostid, and check the license only at server startup.

For this example, you would use a HOST line and license like this. Note that this example uses a "rehostable" hostid. This allows your customer to rehost the license server, but it also allows you to disable the licenses later. In this example, the tolerance (10– the 3rd parameter in the options string) is unused.

```
HOST localhost license=1234
LICENSE isvname rlm_server 1.0 1-jan-2025 uncounted
RLM Reference Manual Page 153 of
296hostid=rehost=cc5a340f:84ca2561da285b7cbac719a916b016dcffde6434
issuer=http://www.hostedactivation.com akey=1234-5678-9999-4321
options=1234:startup:10 signature
```

Example 3. Lock your server to a license with no automatic activation server involvement.

For this example, you would use a SERVER line and license like this. In this example, your customer would activate (and deactivate) this hostid using RLM's web interface, but the license server would not automatically contact the activation server at any time.

```
HOST localhost license=1234
LICENSE isvname rlm_server 1.0 1-jan-2025 uncounted
hostid=rehost=cc5a340f:84ca2561da285b7cbac719a916b016dcffde6434
options=1234:none:0 signature
```

5.23.4 Errors - What happens if the license server can't contact the activation server, or gets an error?

The license server attempts to automatically contact the activation server for either the "startup", "daily", or "6hour" check types. This check may result in an error from the activation server. Depending on when the error happens, it is handled as follows:

For the "startup". "daily", and "6hour" types, the license server needs to check the activation key at startup time and periodically afterwards. The errors from the activation server which are tolerated at startup time are minimal.

At startup time, the following status returns will be ignored, and the server will continue to run if the license is still valid:

```
0 (good status)
RLM_EH_INTERNAL
RLM_EH_NET_RERR
RLM_EH_NET_WERR
RLM_ACT_CANT_WRITE_FULFILL
RLM_ACT_CANT_WRITE_KEYS
RLM_ACT_CANTREAD_DB
```

Any other status return will cause the license server startup to fail, and the server will only process node-locked, uncounted licenses from this license file, and no check with the activation server will be attempted in order to recover.

For "daily" or "6hour" checking, at the re-check, any errors will be tolerated, however it will be considered a failure. If the check fails with an error for "tolerance" number of checks in a row, the server will continue to run, but behave as if any served licenses were removed from the license file.

In all cases, the following errors will cause the server to immediately stop serving licenses locked to this hostid:

```
RLM_ACT_NO_KEY
RLM_ACT_NO_PROD
RLM_ACT_NO_KEY_MATCH
RLM_ACT_KEY_DISABLED
RLM_ACT_KEY_NO_HOSTID
```

5.23.5 Use Case #1: Using Alternate Server Hostids to support rehosting of License Servers

In order to allow your customer to rehost their license server, use the following procedure:

1. In rlm_isv_config.c, be sure that you have set the activation server's URL with the *rlm_isv_cfg_set_url()* call. The default URL is "hostedactivation.com", so if you use Reprise's hosted activation service, you can skip this step. Everyone else needs to set the URL here. None of this will work without the correct URL setting.
2. In rlm_isv_config.c, be sure to include RLM_ACTPRO_ALLOW_ASH in the "allowed_types" variable in the call to rlm_isv_cfg_actpro_allowed_hostids().
3. Set the URL in Activation Pro, in the "Admin"/"Database"/"Activation Server URL" section. This causes the correct URL to be placed into the license.
4. Set up a product in Activation Pro with product name "rlm_server", license type "Alternate Server Hostid". Set the check type and tolerance interval here. Check only "rehostable" under "Allowed Hostid Types". Check "Yes" under "Include Activation Key in License". Leave the "Other RLM keyword=value pairs" field blank – Activation Pro will fill this in.
5. Create an activation key for this product. Saving the activation key will cause ActPro to generate a unique serial number in the "ash_sn" column of the "keyd" table. You should issue a single activation key for a single Alternate Server hostid, and **you should not delete this key if you ever want to get that serial number back again.** Leave the "Other RLM keyword=value pairs" field blank.
6. Give the activation key to your customer.
7. Your customer will need to bring up RLM with a license file that has an ISV line for your ISV server, so that your ISV server will start. They can start RLM with a license file containing these two lines:

```
HOST localhost ANY
ISV your-isvname
```

8. Your customer then goes to the RLM web interface and selects "Status". At the far right-hand side of your ISV server's line, there is a button labeled "Activate" (the user must have the "edit options" privilege for this button to appear). They should click this button, then enter the activation key you supplied (from step 5, above), and press "Activate/Deactivate Alternate Server Hostid". If the operation succeeds, your ISV server will write a license file named "rlm_your-isvname_activation_key.lic" in the current directory (if your customer deactivates a rehostable hostid-based license, this license file will be deleted).
9. Your customer should copy the HOST line from the new license file to their main license file for this server. This can be done with the "Edit License Files" button in the RLM web interface.
10. Next, when your customer activates licenses to be used on this server, they should use the hostid "**license=xxx**" from the HOST line in this license file. (Reprise recommends that you keep the rlm_your-isvname_ash.lic file separate from the file with all your regular licenses, since the Alternate Server Hostid file will be replaced on a re-activation.
11. At this point your customer can deactivate the Alternate Server Hostid license, move the license file and software to another machine, and re-activate it there.

5.23.6 Use Case #2: Remotely Disabling License Servers

In order to be able to disable your customer's license servers, use the same procedure as Use Case #1, with the following exceptions:

- You would always use "startup", "daily", or "6hour" for checking, never "none"
- You can use a rehostable hostid as in Use Case #1, or you can leave the "Allowed hostid types" blank in order to use a standard hostid.

5.23.7 Use Case #3: What do you do if the user loses his license file(s)?

This is somewhat of a sticky situation, but you can recover as follows:

1. Get the ASH license itself:
    - Call rlm_get_rehost() with the product name "rlm_server"
    - Call rlm_act_keyvalid_license() with the ASH activation key and the hostid from the rlm_server license from the previous step.
2. Extract the license # from the ASH license:
    - Write the license from step 1 into a new license file, call rlm_init().
    - RLM_LICENSE lic = rlm_checkout(RLM_HANDLE, "rlm_server", "1.0", 1);
    - char *options = rlm_license_options(lic);
    - char *p = strchr(options, '.'); if (p) *p = '0';
    - sprintf(ash_hostid, "license=%s", options);
3. With the ASH hostid, retrieve the other licenses with rlm_act_keyvalid_license()
4. Write the new license file for the other licenses:
    - SERVER localhost license=ash_hostid#
    - ISV your-isvname

- LICENSE 1
- LICENSE 2
- …
- LICENSE n

Here's some example code:

The routine write_license() is assumed to write the license file with the string specified.

```
  RLM_LICENSE lic;
  char *options = (char *) NULL, *p = (char *) NULL;
  char rehost_hostid[RLM_MAX_HOSTID_STRING+1];
  char ash_hostid[RLM_MAX_HOSTID_STRING+1];
  char license[RLM_MAX_LINE*10];

    i = rlm_get_rehost(rh, "rlm_server", rehost_hostid);
    if (i) { ERROR }
    i = rlm_act_keyvalid_license(rh, acthost,
"activation-key-for-ASH-license",
                                                        rehost_hostid,
license);
    if ( !i )
    {
      rlm_putenv("RLM_LICENSE=”your-ASH-lf");
      write_license(“your-ASH_lf”, license);
    }

#if 0   /* This was the code to use before rlm_act_fulfill_info() was
available */
    rlm_close(rh);
    rh = rlm_init("", argv0, (char *) NULL);
    rlm_putenv("RLM_LICENSE=.");
    lic = rlm_checkout(rh, "rlm_server", "1.0", 1);
    if (lic) options = rlm_license_options(lic);
    if (options) p = strchr(options, ':');
    if (p) *p = '\0';
    if (options) sprintf(ash_hostid, "license=%s", options);
    else ash_hostid[0] = '\0';
    rlm_act_keyvalid_license(rh, acthost,
"activation-key-for-floating-licenses”,
                                                        ash_hostid, license);
#else   /* code to use post v12.1, with rlm_act_fulfill_info() */
  {
    char prod[RLM_MAX_PRODUCT+1], ver[RLM_MAX_VER+1], exp[RLM_MAX_EXP+1];
    int date_based, license_type, upgrade_version, count, fulfilled, rehosts,
revoked;
    /* all vars on above 2 lines unused */

    rlm_act_fulfill_info(rh, acthost, "activation-key-for-ASH-license", prod,
vers, &date_based,
                                &license_type, upgrade_version, &count,
&fulfilled, &rehosts,
                                &revoked, exp, ash_hostid);
  }
#endif
    if ( !i )
    {
      char lic2[RLM_MAX_LINE*10];
```

```
        sprintf(lic2, "SERVER localhost %s\nISV reprise\n", ash_hostid);
        strcat(lic2, license);
        write_license("activation1.lic", lic2);
        reread_server();
    }
```

5.24 Alternate Nodelock Hostids

The *Alternate Nodelock Hostid* capability works in conjunction with RLM Activation Pro, and is not supported when used without ActPro.

5.24.1 Overview

An *Alternate Nodelock Hostid* is very similar to an Alternate Server Hostid, except that it is used to lock a group of licenses to a particular node, and there is no automatic checking of the *Alternate Nodelock Hostid*. With that in mind, the remainder of this chapter is a simplified version of the Alternate Server Hostid chapter.

An *Alternate Nodelock Hostid* is a method to lock one or more individual licenses to a node-locked, uncounted license, rather than directly to a hostid. The main uses for this are to lock the licenses to a rehostable hostid (so your customer can revoke and re-authorized all licenses with a single revoke/re-authorization request pair), or so that you can disable the licenses that are in the field. Also, this allows you to lock multiple independent versions of a particular license to a rehostable hostid, which isn't possible using a rehostable hostid directly.

The *Alternate Nodelock Hostid* hostid type, which has the same syntax as the *alternate server hostid* type (license=serial#) is supported by RLM only as a nodelock hostid on a LICENSE line. A single computer can only use a single *Alternate Nodelock Hostid* locked to a rehostable hostid, i.e., there cannot be two different "license=xyz" hostids in license files if the two *Alternate Nodelock Hostids* both use rehostable hostids.

Note Unlike an Alternate Server Hostid, there is no runtime checking to verify that the activation key used to generate the *Alternate Nodelock Hostid* is still valid. If you want this checking, you will need to call rlm_act_keyvalid() at the appropriate time(s).

The license for the *Alternate Nodelock Hostid* must have the following characteristics. (This is for the *Alternate Nodelock Hostid* license only, these restrictions do not apply to the licenses which are locked to the *Alternate Nodelock Hostid*.) Also note that the options string is generated correctly by Activation Pro when you select the "*Alternate Nodelock Hostid*" license type, so you don't have to worry about that part of the license:

- It must be a nodelocked, uncounted license (ActPro does this automatically when you select the "*Alternate Nodelock Hostid*" license type).
- Product name: rlm_nodelock (you must set the product name correctly in ActPro)
- Options string, as defined below (set automatically by activation pro)
- Serial# must be a decimal number of <= 15 digits (activation pro generates this automatically every time it creates an activation key)
- hostid: required
- akey= recommended if you want to call rlm_act_keyvalid()
- Set issuer to the URL of the activation server (recommended if you want to call rlm_act_keyvalid()

The options string has the following format:

```
options=serial#
```

Where:

serial# - is the serial number from the license=serial# specification on the product's LICENSE line

---

Note The serial number is the part of the hostid that makes the nodelocked licenses in the license file unique. Activation Pro will give each of your customers a different serial number automatically (actually, it gives each activation key a unique serial number, so you should only generate a single Alternate Nodelock Hostid from one activation key).

When creating the product definition for an *Alternate Nodelock Hostid* license, use ActPro's "*Alternate Nodelock Hostid*" License type in the product definition for this – this product definition will generate the LICENSE line required. The serial number is generated automatically when an activation key is created and cannot be changed. You MUST, however, set the product name to "rlm_nodelock".

When creating your product definition, be sure to select "Yes" in the "Include Activation Key in License" parameter if you are going to call rlm_act_keyvalid() later, and if you are using this capability to provide rehostable licenses, select only "rehostable" in the "Allowed Hostid Types" selection of either the product definition or activation key.

The following code sample will create the alternate nodelock hostid, assuming the activation key "anh-key" is an activation key with the properties above:

```
RLM_ACT_HANDLE act_handle;

        act_handle = rlm_act_new_handle(rh);
        rlm_act_set_handle(act_handle, RLM_ACT_HANDLE_REHOST, (void *) 1);
        i = rlm_activate(rh, althost1, "anh-key", 1, license, act_handle);
        rlm_act_destroy_handle(act_handle);
```

Once you have activated the ANH license and written "license" to a local license file, you can get the hostid to use in the other licenses with code like the following:

```
RLM_HANDLE rh;
RLM_LICENSE lic;
char *options;
char *hostid[RLM_MAX_HOSTID_STRING];

        strcpy(hostid, "invalid");

        rh = rlm_init(...);
        lic = rlm_checkout(rh, "rlm_nodelock", "1.0", 1);
        if (!rlm_license_stat(lic))
        {
                options = rlm_license_options(lic);
        }
        if (options) sprintf(hostid, "license=%s", options);
```

Warning If you use alternate nodelock hostids, **you MUST set the URL in the ActPro database** in the admin->database tab.

5.24.2 Alternate Nodelock Hostid Example

Example: Four product licenses locked to an Alternate Nodelock Hostid.

For this example, we show the *Alternate Nodelock Hostid* license and the four product licenses in separate boxes. In practice, you should put the *Alternate Nodelock Hostid* license in a separate license file so that you can easily copy the product licenses to a 2nd machine and abandon the old *Alternate Nodelock Hostid* license when you revoke and re-authorize it with a new one.

```
LICENSE isvname rlm_nodelock 1.0 1-jan-2025 uncounted
hostid=rehost=cc5a340f:84ca2561da285b7cbac719a916b016dcffde6434
options=1234 akey=1234-5678-abce-efgh
issuer=http://hostedactivation.com signature
```

```
LICENSE isvname product1 1.0 1-jan-2025 uncounted
hostid=license=1234 signature
LICENSE isvname product2 1.0 1-jan-2025 uncounted
hostid=license=1234 signature
LICENSE isvname product3 1.0 1-jan-2025 uncounted
hostid=license=1234 signature
LICENSE isvname product4 1.0 1-jan-2025 uncounted
hostid=license=1234 signature
```

5.24.3 Use Case #1: Using Alternate Nodelock Hostids to support rehosting of a group of product Licenses

In order to allow your customer to rehost a group of nodelocked licenses, use the following procedure:

1. In rlm_isv_config.c, be sure to include RLM_ACTPRO_ALLOW_ASH in the "allowed_types" variable in the call to rlm_isv_cfg_actpro_allowed_hostids().
2. Set the URL in Activation Pro, in the "Admin"/"Database"/"Activation Server URL" section. This causes the correct URL to be placed into the license. While this step is optional, the generation of the *Alternate Nodelock Hostid* will have the issuer=URL keyword in the license, which you can use if you want to call rlm_act_keyvalid().
3. Set up a product in activation pro with product name "rlm_nodelock", license type "*Alternate Nodelock Hostid*". Check only "rehostable" under "Allowed Hostid Types". Check "Yes" under "Include Activation Key in License". Leave the "Other RLM keyword=value pairs" field blank – the Activation pro license generator will fill this in.
4. Create an activation key for this product. Saving the activation key will cause ActPro to generate a unique serial number in the "ash_sn" column of the "keyd" table. Note that you should issue a single activation key for a single *Alternate Nodelock Hostid*, and **you should not delete this key if you ever want to get that serial number back again**. Leave the "Other RLM keyword=value pairs" field blank.
5. Give the activation key to your customer.
6. Your customer then activates this license using your activation utility. Store this license in license file #1.
7. Your customer can then activate any nodelocked product licenses using the hostid "license=xyz", where xyz is the serial number of the license generated in step 6. It is best practice to keep these licenses in a different license file from the one created in step 6, since the *Alternate Nodelock Hostid* license file will be replaced on a re-activation.
8. At this point your customer can deactivate the *Alternate Nodelock Hostid* license, move the license file and software to another machine, and re-activate it there. The individual

product licenses do not need to be re-activated.

5.24.4 Use Case #2: Remotely Disabling Nodelocked Licenses

In order to be able to disable your customer's nodelocked licenses, use the same procedure as Use Case #1, with the following exceptions:

- Your code would call rlm_act_keyvalid() to verify that the activation key was not disabled. When you want to disable the licenses, disable the activation key in ActPro.
- For this case, you can use a rehostable hostid as in Use Case #1, or you can leave the "Allowed hostid types" blank in order to use a standard hostid for the Alternate Nodelock Hostid.

5.25 Dynamic Reservations

At certain times, it is desirable for one process to check out a license and ensure that a new process can get the same license. In order to accomplish this, RLM introduced the concept of a *dynamic reservation*.

While there may be many use cases for this within an ISV's licensing design, the most general usage is in the case of *job schedulers*, such as IBM's LSF and Runtime Design Automation's WorkloadXelerator.

Dynamic reservations are accomplished with no changes to the RLM API - all capabilities are controlled via the environment variables RLM_RESERVE and RLM_RECLAIM. Of course, these variables can be set with the RLM API call rlm_setenv(). The reason we use environment variables for this capability is so that the job scheduler can set the environment before launching the ultimate application, and each ISV does not have to write code to specifically handle the dynamic reservation. Once linked with RLM v12.2+, every RLM-licensed application is enabled to use this capability.

To use a dynamic reservation, we will refer to the two processes as the reserving process – i.e., the process which creates the reservation, and the reclaiming process – the process which reclaims the license which was previously reserved.

5.25.1 How to use Dynamic Reservations

Creating the Reservation

In the reserving process, set RLM_RESERVE to:

```
isvname:#secs:string
```

Where:

isvname – is the name of the ISV which "owns" the license. If left blank, the current ISV name is used. If specified and different from the current ISV, then the checkout request will not perform encryption handshaking with the license server.

#secs – is the number of seconds to "hold" the reservation after the license is checked out by the reserving process.

string – is the string, or handle, that the reclaiming process must use to reclaim this license (max 32 bytes).

Once the license is checked out, the reserving process should check it back in, and start

the reclaiming process within *#secs* seconds. Note that if the license itself specifies a hold time or a minimum_checkout time that time will apply to the checkout *if the isvname is blank or the same as the current ISV. If the isvname is different from the current ISV, no hold time or minimum_checkout time will apply to the checkout itself.*

If the ISV requesting the dynamic reservation is the same as the ISV that "owns" the licenses:

- RESERVED, OUT and IN will all be logged in the debug log.
- The new DYNRES line, OUT and IN will be logged in the report log.

If the ISV requesting the dynamic reservation is different:

- Only the RESERVED line will be logged in the debug log.
- Only the DYNRES line will be logged in the report log.

Other notes for dynamic reservations:

1. Dynamic reservations are only supported on a license server (ie, not for node-locked, unserved licenses).
2. If you ask to reserve N licenses, the server must have N free licenses available. Shared licenses do not count in this number, and other (non-dynamic) reservations do not count, either. In other words, the server does not consider licenses available for sharing or any reservations when computing the number of free licenses.
3. You cannot QUEUE for dynamic reservations.
4. Dynamic reservations do not work with Roaming or client caching.
5. if both RLM_RESERVE and RLM_RECLAIM are set, RLM_RECLAIM takes precedence, in other words, the reservation will not be created.

5.25.2 Reclaiming the Reserved Licenses

In the reclaiming process, set RLM_RECLAIM to:

```
isvname:#secs:string
```

Where:

isvname – is the name of the ISV which "owns" the license. This will normally be blank, but in the case of a job scheduler which needs to cancel a reservation it created earlier, is will be filled in. If left blank, the current ISV name is used. If specified and different from the current ISV, then the checkout request will not perform encryption handshaking with the license server.

#secs – is the number of seconds to "hold" the reservation after the license is checked back in by the reclaiming process.

string – is the string, or handle, to reclaim the license.

Once the license is checked out, the reclaiming process can use it as normal, and at the end, the reservation will be released in #secs seconds. The license checkout is subject to the same rules as for the reserving process, i.e., if the license itself specifies a hold time or a minimum_checkout time that time will apply to the checkout *if the isvname is blank or the same as the current ISV. If the isvname is different from the current ISV, no hold time or minimum_checkout time will apply to the checkout itself.*

The reservation created by the reserving process will be independent of all the license restrictions specified in the license itself. So, for example, if that license is shareable by user/host, then the reclaiming process will be able to share the resulting license with other processes started on the same user/host. In this case, only the first process needs to present the handle to the server, the others can share that license without setting the RLM_RECLAIM environment variable.

Also note that once the reservation is created, multiple sequential processes can reclaim the same reservation with the RLM_RECLAIM environment variable. The last process which is going to use the reservation should set #secs to 0, so that the reservation goes away at that point. If a job scheduler creates a reservation, then later discovers that the jobs cannot continue to run, the scheduler can remove the reservation by setting RLM_RECLAIM with #secs set to 0.

Some notes and restrictions:

1. Dynamic reservations are only supported on a license server (ie, not for node-locked, unserved licenses).
2. You cannot QUEUE for dynamic reservations.
3. Dynamic reservations do not work with ROAMING.
4. If both RLM_RESERVE and RLM_RECLAIM are set, RLM_RECLAIM takes precedence.
5. License sharing is never considered when creating dynamic reservations.
6. Personal licenses and Dynamic reservations use the same variables for transport, so it is not possible to have a personal license that uses dynamic reservations. This is a minor restriction, since the use cases for these 2 kinds of licenses are so completely different.

Note Dynamic reservations are not supported on Solaris.

5.26 Using RLM with HTTPS

Beginning in RLM v14.0, your RLM-enabled application can communicate with an RLM Cloud license server using HTTPS, rather than the native RLM socket protocol. Beginning in RLM v14.1, Activation Pro supports the HTTPS protocol as well.

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 RLM Cloud servers or Activation Pro, you can ignore this chapter.

5.26.1 How to use HTTPS with RLM Cloud or 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_WS_NOSUPP when web services is specified in the license file, or RLM_EH_NOHTTPSSUPPORT if you specify an HTTPS URL to Activation Pro (or your URL redirects to an HTTPS URL).

For RLM Cloud, to include HTTPS support in your application, follow these instructions:

- Be sure to set the appropriate value in the rlm_isv_cfg_set_promise() call contained in rlm_isv_config.c. By default, this is 10 minutes, which means you need to call rlm_get_attr_health() every 5-7 minutes.
- Call rlm_isv_cfg_set_isv_handshake() in rlm_isv_config.c with an appropriate set of P1 and P2 parameters (please do not use the defaults values in rlm_isv_config.c).

For both RLM Cloud and Activation Pro, follow these (additional) 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 exampleclient_http.exe in the makefile for an example.
  - If using a DLL, link the appropriate rlm_msgs_http.obj with the DLL and then link only the include library with your application. see the DLL and exampleclient_http_md.exe rules 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.

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:
For RLM Cloud:

Change the normal RLM Cloud CUSTOMER line to use the HTTPS port. So, for a normal RLM Cloud license file, there would be a CUSTOMER line like the following:

```
CUSTOMER customer-name isv=isvname server=lsN.rlmcloud.com port=5053
password=XXX
```

To use the HTTPS transport, simply change the port number from 5053 to 443:

```
CUSTOMER customer-name isv=isvname server=lsN.rlmcloud.com port=443
password=XXX
```

For Activation Pro:

Specify 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.

5.26.2 What if I I use RLM Cloud and don't build in web services support

If you use RLM Cloud and don't build in web services support, you must be prepared to receive the RLM_EH_WS_NOSUPP error from various RLM functions (or RLM_EH_NO-HTTPSSUPPORT for the rlm_activate() family of functions). If your customer changes the RLM CUSTOMER line to use port 443, any requests you make thru the RLM client library will return RLM_EH_WS_NOSUPP.

If you use web services, you might get a RLM_EL_BADHANDSHAKE error if a checkout succeeds and the handshake is bad. In this case, you should be sure to check the license back in, otherwise it will remain checked out on the server until the promise interval expires.

Note Calling rlm_checkin() after a failed checkout is always a fine thing to do.

5.26.3 Some notes on MacOS

Earlier versions of MacOS have a bug in the CURL libraries and HTTPS will not work. What we know for sure at this point is that 10.5.8 (Leopard) and earlier do not work, and that 10.12 (Sierra) and presumably later do work.

5.26.4 Unsupported RLM API functions with web services

When using the HTTPS transport, there are RLM API functions which are not supported, and if you call any of these, your code should be prepared to receive an RLM_EH_WEBS_NOSUPP error, or the function may simply be ineffective These functions continue to work for local license files, and license files which specify a server that does not use web services, they just don't work on any license file in your path that specifies web services access to the server. These functions are:

- Licenses cannot be roamed when using web services.
- You cannot queue for licenses with web services – RLM_QUEUE is ignored.
- rlm_auth_check() - used to detect a hacked server, which is not an issue for RLM Cloud.
- rlm_set_attr_logging() is ignored.
- Dynamic reservations can only be created/reclaimed for the same ISV name. This means that if you want a job scheduler to create dynamic reservations for you, you will have to write a small utility program to create the reservation and use non-HTTPS transport.
- Client timezone= and platforms= license keywords are unused – the timezone will always appear to be the servers timezone, and the platform will always be x64_l.
- The "disallow_generic" server setting and the "isv string invisible" settings are ignored.
- The following client environment parameters are not passed to the server: RLM_PROJECT, client computing environment, client os version, client's argv[0]
- Client authentication with rlmadduser is not supported with HTTPS.

5.26.5 Using RLM/HTTP with proxy servers

RLM uses CURL for https transport. If you have a proxy server, you will need to set the CURL environment variable http_proxy to the address of your proxy server. See the CURL documen-

tation.documentation.

5.26.6 RLM Cloud Performance Metrics

We have run tests on both native and web services based versions of RLM, from various locations (always to an RLM Cloud license server in the Dallas datacenter). The test performs 100 rlm_init()/rlm_checkout()/rlm_checkin()/rlm_close() calls and takes the average time (all in milliseconds). The "location" column indicates where the client software is running. The results are summarized in this table:

Client Location	Checkout time, native	Checkout time, web services
Dallas (same as server)	10-20 msec	280 msec
San Francisco	570 msec	610 msec
New York	280 msec	460 msec

And using a loop of 100 checkout/check-ins (without rlm_init()/rlm_close()):

Client Location	Checkout time, native	Checkout time, web services
Dallas (same as server)	< 10 msec	270 msec
San Francisco	330 msec	600 msec
New York	190 msec	

5.26.7 Performance Comparison

If you want to understand why web services for RLM Cloud is so much slower, this section will give you an idea. With the native comm package, the following happens when you check out a license:

1. Send a message to RLM on the server host, get the address of the ISV server (this is done once per customer name on a particular client machine, and the result is cached).
2. Send a "hello" message to the ISV server and read the response.
3. Send a "checkout" message to the ISV server and read the response.
4. Send a heartbeat message to the ISV server and read the response later.

With web services, it is a lot more complicated:

1. Your program makes an HTTPS request to the license server on the server host.
2. The Apache web server on that host starts a php program to process the request.
3. The php program starts a helper program – (which is a lot like your program would have been if you used the normal comm package) - via another http request to the web server, which then:
    - Sends a hello message to the ISV server, reads the response.
    - Sends a "checkout" message to the ISV server, reads the response.
    - Sends a heartbeat message to the ISV server, it will read the response later.
    - Formats the response back to the php program that started it.
4. Now, the php program re-formats that data in json and returns it, via the Apache web server, to the web services client (i.e., your program)
5. Your program parses the json and returns the status to you.

6. Next, when you call rlm_get_attr_health(), you repeat steps 1-5 again, rather than just reading a response that was already waiting on a socket that was already open.

This brings up another important point. In the normal RLM comm package, heartbeats are "primed" and the response is not checked until later, so the application never waits for a round-trip to the server for a heartbeat. In web services, the heartbeat is always synchronous, so the whole round-trip overhead is incurred on each heartbeat. So don't do them too often.

Reference Material

6.1 Appendix A – RLM API

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

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

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

6.1.1 RLM_HANDLE

RLM_HANDLE is the main handle in RLM. 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 API calls.

6.1.2 RLM_LICENSE

RLM_LICENSE is the license handle in RLM. 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.

6.1.3 Core API

These 8 functions provide all basic licensing operations needed for most applications:

Function	Description
rlm_init(), rlm_init_disconn()	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 by checking server.

6.1.4 Advanced API

Most applications will need few, if any, of these calls:

Function	Description
rlm_act_request()	Request license activation from the internet.
rlm_activate()	Request license activation from the internet.
rlm_act_new_handle(), rlm_act_destroy_handle()	Create/destroy handle to pass activation parameters.
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.
rlm_act_keyvalid(), rlm_act_keyvalid_license()	Verify that an activation key still has a valid license on this hostid from the activation server.
rlm_act_refresh()	Request refreshing of all refreshable licenses. [Deprecated]
rlm_act_rehost_revoke()	Revoke a rehostable license.
rlm_act_revoke_disconn()	Revoke a disconnected rehostable license.
rlm_act_set_handle()	Set data in activation handle.
rlm_add_meter_count()	Add count to a specified meter.
rlm_auth_check()	Ask the server to verify a license.
rlm_auto_hb()	Enable Automatic Heartbeats.
rlm_set_auto_hb_isvdata()	Enable Automatic Heartbeats.
rlm_add_meter_count()	Add count to a specified meter.
rlm_checkout_product()	Request an exact license from RLM.
rlm_detached_demo()	Install RLM Detached Demo™ license.
rlm_detached_demox()	Remove RLM Detached Demo™ license.
rlm_errstring_num()	Translate RLM status value into a string.
rlm_get_attr_lfpath()	Get license path in use by RLM.
rlm_get_rehost()	Retrieve the hostid of a rehostable license.
rlm_hostid()	Retrieve the hostid of this machine.
rlm_all_hostids()	Retrieve the hostid of this machine.
rlm_all_hostids_free()	Retrieve the hostid of this machine.
rlm_license_XXXX()	Get checked-out license information.
rlm_log()	Log ISV-specific data.
rlm_dlog()	Log ISV-specific data.
rlm_products()	Generate list of products that can be checked out.
rlm_putenv()	Set environment variable within the application.
rlm_set_active()	Inform RLM of activity status of application.
rlm_set_environ()	Set user/host/ISV-defined values for RLM.
rlm_set_attr_keep_conn()	Set "keep connection" status for RLM.
rlm_set_attr_logging()	Turn server logging on or off.
rlm_set_attr_password()	Set license password-string for future.
rlm_set_attr_req_opt()	Set required substring in license options.
rlm_set_attr_reference_hostid()	Set reference hostid for ActPro.
rlm_sign_license()	Sign an individual license in-memory.
rlm_skip_isv_down()	Enable "skip" of license servers where your ISV.
rlm_temp_xxx()	Temporary license functions.
rlm_act_revoke()	*Deprecated.*
rlm_act_revoke_reference()	*Deprecated.*

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

---

rlm_act_request()

Request license activation from the internet

Note New code should use the *rlm_activate()* call instead of *rlm_act_request()*. *rlm_act_request()* will be supported in the future, but all new options will be added only to *rlm_activate()*.

```
#include "license.h"
RLM_HANDLE rh;
int stat;
const char *url;
const char *isv;
const char *akey;
const char *hostid_list;
const char *hostname;
int count;
const char *extra;
char license[RLM_ACT_MAX_LICENSE+1];

rh = rlm_init(...);
stat = rlm_act_request(rh, url, isv, akey, hostid_list, hostname, count,
extra, license);
```

rlm_act_request() requests a license activation from the server at *url* for ISV name *isv*. The activation key *akey* and license count *count* are sent to the server.

Note If the license count is <= 0, a count of 1 is used.

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 either the *akey* or *extra* parameter contains an embedded newline, *rlm_act_request()* will return RLM_EL_BADPARAM.

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 rebuilding 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 The URL should **always** be http, never https. rlm_act_request() encrypts the request independent of the webserver.

---

hostid_list is a space-separated list of RLM standard hostids. If hostid_list is NULL or empty, the hostid list from the current machine is used. Note that ethernet addresses should be exactly 12 hex characters (without the leading "0x"), and 32-bit hostids should be hex numbers without the leading "0x". RLM will select a hostid from this list for use in the license, as described in the next paragraph.

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_act_request()*. 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 Section 4.3.8 for more details.

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

- rehostable hostid
- ISV-defined hostid
- ISV string hostid
- rlmid hostid
- Disk Serial Number
- ethernet address
- UUID
- 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. 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 allowed hostid types from the license string, use code similar to this:

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

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 *hostname* is NULL or empty, the hostname of the current machine is used.

extra is a string containing extra "keyword=value" license attributes. These must be valid RLM license syntax, not just any keyword=value pair.

Note The extra 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 spaceseparated 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.

The parameter *license* must be an allocated string of length RLM_ACT_MAX_LICENSE+1. If *rlm_act_request()* 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.

Error Codes

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 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_act_request()*.

rlm_act_request() encrypts the data sent to the activation server. 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 Section 6.1.

---

rlm_activate()

Request a license activation from the internet.

```
#include "license.h"
RLM_HANDLE rh;
int stat;
const char *akey;
int count;
char license[RLM_ACT_MAX_LICENSE+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 If the license count is <= 0, a count of 1 is used.

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 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 generated.

If the *akey* parameter contains an embedded newline, *rlm_act_request()* will return RLM_EL_BADPARAM.

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 rebuilding 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 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_act_request()*. 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 Section 4.3.8 for more details.

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

- rehostable hostid
- ISV-defined hostid
- ISV string hostid
- rlmid hostid
- Disk Serial Number
- ethernet address
- UUID
- 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. 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 RLM_ACT_MAX_LICENSE+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.

Error Codes

See Error Codes above.

Proxy Server Support

See Proxy Server Support above.

Back to Section 6.1.

---

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);**

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);**
```

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

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 The URL should **always** be HTTP, **never** HTTPS. rlm_act_info() encrypts the request independent of the webserver.

Note 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 error 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 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. This date is returned in the ActPro sortable date format of yyyy-mm-dd, rather than the RLM standard date format of dd-mmm-yyyy.
- **hostid** – The hostid from the latest fulfillment.

In addition to the above, **rlm_act_keyinfo2()** also returns:

- **keyexp** – The expiration date of the activation key itself. This date is returned in the ActPro sortable date format of yyyy-mm-dd, rather than the RLM standard date format of dd-mmm-yyyy.

- **allowed_hostids** – The bitmap of allowed hostids for this activation key.
- **sub_interval** – The subscription interval, as follows:
    - -1: monthly subscription
    - -2: quarterly subscription
    - -3: annual subscription
    - >0: # of days in subscription interval
- **sub_window** – The subscription window, in days

Note Unless you are doing offline activation, all rlm_actXXX() calls (except rlm_act_XXX_handle()) require internet access.

Back to Section 6.1.

---

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. This call is intended to be used for nodelocked licenses only. If you want to invalidate a floating license, use Section 5.23.

Note 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. 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()* requires an RLM Activation Pro server to return the license.

Note Unless you are doing offline activation, all rlm_actXXX() calls (except rlm_act_XXX_handle()) require internet access.

Back to Section 6.1.

---

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.

Note Unless you are doing offline activation, all rlm_actXXX() calls (except rlm_act_XXX_handle()) require internet access.

Back to Section 6.1.

---

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 license-rehosting 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. *val* is a string containing extra "keyword=value" license attributes. These must be valid RLM license syntax, not just any keyword=value pair.

> **Warning** 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 is also 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
```

OR

```
list-of-hostids
```

For example:

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

OR

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

What is the difference between using "list:" and not?

- With a leading "list:", the hostid list is used without checking whether the hostids are allowed or not. For nodelocked licenses, the whole list will be in the license, meaning the license will work on any hostid in the list.
- Without the leading "list:", each hostid in the list will be checked against the allowed hostid types, and the "most secure" one present will be used. This is the default way that RLM passes client hostids to the activation server if you do not set RLM_ACT_HANDLE_HOSTID_LIST.

If a list is supplied with a leading "list:", 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.

> **Warning** 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. This string cannot contain the '>', '<', or '&' characters.

*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.

> **Warning** Once created, the contents of the rehostable hostid directory **CANNOT BE TOUCHED, MODIFIED, ** **DELETED, or RESTORED from a BACKUP without invalidating the hostid**.

REHOSTable hostids can be used with nodelocked, uncounted, and SINGLE licenses only.

*RLM_ACT_HANDLE_CONTACT: (char * val)*

*RLM_ACT_HANDLE_COMPANY: (char * val)*

*RLM_ACT_HANDLE_EMAIL: (char * val)*

*RLM_ACT_HANDLE_STATE: (char * val)*

*RLM_ACT_HANDLE_COUNTRY: (char * val)*

*RLM_ACT_HANDLE_U1: (char * val)*

*RLM_ACT_HANDLE_U2: (char * val)*

These 7 parameters, all of which take a (char *) value, are used to set the contact and company of the person doing the activation. If any of these parameters are set, CONTACT and COMPANY must both be set, otherwise you will receive an RLM_AT_CONTACT_BAD error from rlm_activate(). Upon activation, this information will populate the contact and company tables.

Back to Section 6.1.

---

rlm_act_refresh()

Request refreshing of all refreshable licenses.

> **Warning** rlm_act_refresh() is deprecated in RLM v9.3, in favor of the new rehostable licenses.

```
#include "license.h"
RLM_HANDLE rh;
char \*isv;
void (*callback)(char \*licfile, char \*license, int status);

rh = rlm_init(...);
error_count = rlm_act_refresh(rh, isv, callback);
```

rlm_act_refresh() requests refreshing of all the refreshable licenses known to rlm through the paths supplied to rlm_init() when *rh* was created, plus licenses pointed to by the RLM_LICENSE and <isv>_LICENSE environment variables, if set. The act of refreshing the license involves a transaction with the activation server.

isv is your ISV name. *callback* is a function supplied by the ISV that is called after each individual license refresh is attempted. *licfile* is the name of the license file containing the rehostable license, *license* is the name of the license on which a refresh was attempted, and *status* is the status of the refresh attempt. Values of *status* are as for *rlm_activate()/rlm_act_request*, except that a status of 1 means that the license was refreshed earlier the same day. *callback* may be specified as NULL. If specified, *callback* is invoked for each successful license refresh as well as unsuccessful ones. *error_count* is the number of licenses where the refresh attempt failed.

Back to Section 6.1.

---

rlm_act_rehost_revoke()

Revoke a rehostable license.

rlm_act_revoke_disconn()

Revoke a disconnected rehostable license.

```
#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);
```

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, and 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 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 license-rehosting.

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.

Note 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 Section 6.1.

---

rlm_add_meter_count()

Add count to a specified meter.

```
#include "license.h"
int status;
RLM_HANDLE rh;
RLM_PRODUCTS product;
int counter, count;

status = rlm_add_meter_count(rh, product, counter, count);
```

rlm_add_meter_count() is used by an application to add *count* to a server's meter *counter*.

The *rlm_add_meter_count()* call requires a connection to a license server which was created with rlm_init() (NOT rlm_init_disconn()). The license server connection is specified in one of 2 ways:

- If a product is specified via the product and rh parameters, this connection is used.
- If no product is specified, the connection from the supplied RLM_HANDLE is used. This connection must have been established prior to the call to rlm_add_meter_count() via an rlm_checkout() call.

If no server connection is specified:

- If neither a product nor an RLM_HANDLE is specified, RLM_EH_NOHANDLE is returned to the caller.
- If the specified product or RLM_HANDLE does not have a server associated, RLM_E-H_NOSERVER is returned to the caller.

Given a server as specified above, this call will add *count* to the specified meter *counter*.
Example (add 100 to counter for "your-product-name"):

```
int status, counter, count_to_add;
RLM_PRODUCTS products;

  products = rlm_products(rh, "your-product-name", "");
  if (products && rlm_product_ismetered(products))
  {
    counter = rlm_product_meter_counter(products);
```

```
    count_to_add = 100;
    status = rlm_add_meter_count(rh, products, counter, count_to_add);
  }
```

alternately:

```
int status, counter, count_to_add;
RLM_LICENSE lic;

  lic = rlm_checkout(rh, "your-product-name", "1.0", 1);
  if (!rlm_license_stat(lic) && rlm_license_ismetered(lic))
  {
    counter = rlm_license_meter_counter(lic);
    count_to_add = 100;
    status = rlm_add_meter_count(rh, (RLM_PRODUCTS) NULL, counter,
count_to_add);
  }
```

Back to Section 6.1.

---

rlm_auth_check()

Ask the server to verify a license.

```
#include "license.h"
int status;
RLM_LICENSE license;
char \*license_to_check;

status = rlm_auth_check(license, license_to_check);
```

rlm_auth_check() is used by an application to verify the integrity of the license server. This call is not required by RLM and is used only as an additional check that the license server has not been modified.

The *rlm_auth_check()* call asks the license server to verify a signed license (license_to_check) after you have checked out some other license. You pass a node-locked uncounted license to this call as the *license_to_check* parameter. This license can be either a good or a bad license, and you should pass both types at different times to make spoofing this call harder.

> **Warning** Do not include HOST or ISV lines in this license, only the LICENSE line. Also note that these licenses must be node-locked, uncounted (or SINGLE) licenses, and do not enclose the license in angle brackets - "<" and ">".

You must pass a valid RLM_LICENSE handle in, and this handle should represent a checked-out license from a license server. *license_to_check* is a signed license, either valid or not.

This function will return either:

- 0 - the server authenticated the license, and it is correct.
- RLM_EL_SYNTAX (-26) – the license to be checked has a syntax error, or the received message is incorrect.
- RLM_EL_BADKEY (-5) - the license did not authenticate in the server.
- RLM_EH_NOSERVER (-112) - the RLM_LICENSE handle passed in does not have a license checked-out on a license server.

See the Securing Your Application FAQ for more information on how to secure your application and license server.

Back to Section 6.1.

---

rlm_auto_hb(), rlm_set_auto_hb_isvdata()

Enable Automatic Heartbeats.

```
#include "license.h"
RLM_HANDLE rh;
int period;
int auto_reconnect;
void (*notify(RLM_HANDLE, RLM_LICENSE, int, void *));
const void *isv_data;

status = rlm_auto_hb(rh, period, auto_reconnect, notify);
rlm_set_auth_hb_isvdata(rh, isv_data);
```

With RLM, you must either manually send heartbeats to the license server by calling *rlm_get_attr_health()* periodically in your code, or tell RLM to do this for you. *rlm_auto_hb()* is the function that sets up automatic heartbeats. You should call either *rlm_get_attr_health()* or *rlm_auto_hb()*, but not both. Also note that while this is not strictly a requirement, you need to send heartbeats to the server if you want to detect license loss. On the other hand, if your application runs for less than a minute or so, there is no real need to send heartbeats to the license server and you can safely ignore heartbeats.

You can call *rlm_auto_hb()* any time after calling *rlm_init()*. If *rlm_auto_hb()* returns a 0 status, the handler was installed without error, otherwise, status contains the RLM error code.

The first parameter is the handle on which to perform automatic heartbeats. RLM will perform heartbeats on all checked-out licenses on this handle. You can call *rlm_auto_hb()* any time after the *rlm_init()* and before the *rlm_close()* call. In practice, you should call *rlm_auto_hb()* within 30 seconds or so of your first license checkout, but it can be called before any checkouts if you wish.

The second parameter, *period*, is the period at which the heartbeat thread runs. *period* must be at least 2 seconds (and is set to 2 if specified as a lower value). RLM will not send heartbeats any more often than once every 30 seconds even if you set *period* to a value less than 30, however, when the license is lost, rlm_auto_hb() will attempt to re-acquire it every *period* seconds. Also note that there is an interaction between the *period* you pick and the minimum heartbeat interval. For example, if you set *period* to 25 seconds, the heartbeat will not be sent the first time requested (since it is less than 30 seconds), but it will be sent the 2nd time – in other words, after 50 seconds. So setting *period* to a value <30 seconds may cause the heartbeat interval to be longer than 30 seconds.

Note The *period* parameter is set to a minimum of 2 seconds internally.

The third parameter, *auto_reconnect*, is set to 0 if you do not want RLM to attempt to re-acquire lost licenses. If you would like RLM to re-acquire licenses that have been lost, set *auto_reconnect* to one. If RLM does an automatic re-acquisition of the license, your original RLM_LICENSE handle will reflect the new status of the license.

Note If you have modified the ISV data via a call to rlm_set_environ() between the time you made your original checkout and the time of the reconnection, the ISV data sent on the new checkout request will be the new version of the ISV data, NOT the ISV data that was transmitted originally.

The fourth parameter is a function pointer which is called when a license is lost, **after** re-acquisition is attempted. If you supply a NULL pointer in this argument, no callback is done to your code. This function is called independent of the setting of *auto_reconnect*. The *notify* handler will be called as long as the license is lost, and, if you have a reconnection handler set, one time

when license re-acquisition is successful (i.e., when status==0). Note that the last call will not happen if you do not have a reconnection handler installed. The function is called as follows:

```
(*notify)(rh, license, lic_status, isv_data);
```

The *license* parameter is the license which was lost. Note that the status may be 0 when this routine is called if you have set *auto_reconnect* to one and the license has already been re-acquired. This function is called in the context of the thread in which heartbeats are performed. The only RLM functions you can call in this routine are all the *rlm_license_xxxx()* functions **with the exception of rlm_license_stat()**. The third parameter, *lic_status*, is the status of the license (updated after the reconnection attempt). The fourth parameter is a void * that you specified in the latest call to *rlm_set_auto_hb_isvdata()* on this handle, or NULL if rlm_set_auto_hb_isvdata() was not called on this handle. This argument can be used to pass any data from the application to the reconnection handle.

rlm_set_auth_hb_isvdata() specifies the data item to be passed to the automatic heartbeat reconnection function, above.

If you call *rlm_auto_hb()*, heartbeats will be sent to the server whenever your application is running, whether you are actively processing or not. This means that an application which is simply waiting for user input will continue to send heartbeats and will not appear to be inactive to RLM. If you would like your customers to be able to time-out the licenses from applications which are in such an idle state, use the rlm_set_active() call to inform RLM when your application is active or inactive.

Note rlm_auto_hb() uses pthreads on Unix, and Windows threads on Windows.

If you call *rlm_auto_hb()* on Unix, you must include -lpthreads in your link command for your application. In addition to this, on Solaris, you must include -lrt in your link command. On Unix systems, *rlm_auto_hb()* sets the handler for SIGPIPE to SIG_IGN in both the main thread and in the thread that generates the heartbeat.

Note If a license expires, the license server does not invalidate or remove any instances of that license which are checked out at the time of license expiration. Heartbeats on such a license will continue to succeed.

Some additional notes on heartbeats and server status checking

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 Section 6.1.

---

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.

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 *rlm_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 check-ins as well as to tell the server that you're are done with the handle.

Back to Section 6.1.

---

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.

Note If RLM_ROAM is set, the setting of RLM_QUEUE is ignored, ie, you cannot queue for the license.

To get the status of the rlm_checkout call, use rlm_license_stat(license). For a list of status returns, see Section 6.2.

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 Section 5.5.6).
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 Section 6.1.

---

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 from 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 Section 6.2.

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 Section 5.5.6).
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 Section 6.1.

---

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.

Note If you plan to check any licenses in then close the handle (i.e., if you are not going to use the handle after checking a license in), then you should omit the *rlm_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 you're 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

```
rlmssl_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 Section 6.1.

---

rlm_create_temp_license(), rlm_return_temp_license(), rlm_temp_new_handle(), rlm_temp_destroy_handle(), rlm_temp_set_handle()

Temporary license functions

```
#include "license.h"
RLM_HANDLE rh;
RLM_TEMP_HANDLE temp_handle;
const void *val;
int status, what;

(void) rlm_temp_destroy_handle(temp_handle);

temp_handle = rlm_temp_new_handle(rh);

status = rlm_temp_set_handle(temp_handle, what, val);

status = rlm_create_temp_license(rh, temp_handle);

status = rlm_return_temp_license(rlm_license);
```

rlm_create_temp_license() will create a new temporary license specified by the information in *temp_handle*.

rlm_return_temp_license() removes the temporary license and returns it to the source.

rlm_temp_new_handle() creates a new temporary license handle, *rlm_temp_set_handle()* sets the parameters in the handle, and *rlm_temp_destroy_handle()* removes the handle.

rlm_temp_new_handle() creates a blank handle to pass temporary license parameters to *rlm_create_temp_license()*. *rlm_temp_new_handle()* returns a NULL handle error. Call *rlm_temp_new_handle()* before calling *rlm_temp_set_handle()*. After the temporary license has been created, call *rlm_temp_destroy_handle()* to free the memory associated with the handle.

To create a temporary license, first create a handle using *rlm_temp_new_handle()*. *rlm_temp_new_handle()* sets the following defaults:

- RLM_TEMP_SOURCE is set to RLM_TEMP_RLMCLOUD
- RLM_TEMP_TYPE is set to RLM_TEMP_UNCOUNTED

Next, set the parameters using *rlm_temp_set_handle()* as follows:

RLM_TEMP_HANDLE_DURATION (int)	# of minutes until license expires.
RLM_TEMP_HANDLE_NEW_DURATION (int)	New duration at refresh (see below)
RLM_TEMP_HANDLE_SOURCE (integer)	The license comes from the source specified, which is either RLM_TEMP_RLMCLOUD or RLM_TEMP_ACTPRO. Currently only RLM_TEMP_RLMCLOUD is supported; this is the default.
RLM_TEMP_HANDLE_LICENSE (RLM_LICENSE)	A checked-out license. Used for RLM_TEMP_RLMCLOUD only. The RLM_LICENSE must come from an RLM Cloud server, not a local server.
RLM_TEMP_HANDLE_TYPE (int)	Either RLM_TEMP_UNCOUNTED or RLM_TEMP_HANDLE_SINGLE to create an uncounted or single license, respectively. RLM_TEMP_HANDLE_UNCOUNTED is the default.
RLM_TEMP_WINDOW (int)	Refresh window, in seconds, if ==0, no automatic refresh (see below).
RLM_TEMP_AKEY (const char *)	Activation key for activation pro. Unused.
RLM_TEMP_COUNT	# of licenses requested. Unused.
RLM_TEMP_URL (const char *)	URL of the ActPro server. Unused.

Once the license is created, on each subsequent checkout, if the license expires within *window*, minutes, then the license will be renewed for *new_duration* minutes (RLM will do this using this same call, with *new_duration* substituted for *duration*). When RLM performs this automatic renewal, it uses the same call, and if the renewal fails, it returns the original license. If the renewal succeeds, the new license is returned.

Note The new expiration is computed from the time of the renewal, NOT from the time of the initial expiration. Also note that if *window* is specified as 0, no automatic renewal will be done.

For example, the following code will create an uncounted license for "prod" which lasts for 3 days, and which auto-renews when you check out the license 12 hours before the end time for an additional 2 days:

```
#define HOURS 60
#define DAYS (24 * HOURS)
RLM_LICENSE lic;
RLM_HANDLE rh;
rh = rlm_init(....);
lic = rlm_checkout(rh, …..);
if (!rlm_license_single(lic) && !rlm_license_uncounted(lic))
{

  temp_handle = rlm_temp_new_handle(rh);
  if (!temp_handle) /* whatever error code you want */
  rlm_temp_set_handle(temp_handle, RLM_TEMP_DURATION, (void *) 3*DAYS);
  rlm_temp_set_handle(temp_handle, RLM_TEMP_WINDOW, (void *) 12*HOURS);
  rlm_temp_set_handle(temp_handle, RLM_TEMP_NEW_DURATION, (void *) 2*DAYS);
  rlm_temp_set_handle(temp_handle, RLM_TEMP_LICENSE, (void *) lic);
  status = rlm_create_temp_license(rh, temp_handle);

}
```

If *rlm_create_temp_license()* returns 0 status, the license was written and is ready for checkout.

Otherwise status contains the RLM error status.

rlm_return_temp_license() can only be called on a license that is checked out. To return this license, call *rlm_return_temp_license()* with the RLM_LICENSE handle returned by *rlm_checkout()*. If *rlm_return_temp_license()* returns good status (0), the license is checked in, and the handle should not be used again.

For example, to return the license created above:

```
lic = rlm_checkout(rh, "prod", "1.0", 1);
if (lic)
{
status = rlm_return_temp_license(lic);
if (status) rlm_checkin(lic);
}
```

Note If *rlm_create_temp_license()* returns RLM_EH_NOTEMPFROMLOCAL status, this means there was already a temporary license present on the system. In this case, the temporary license might have been refreshed as part of the operation.

Back to Section 6.1.

---

rlm_detached_demo()

Install RLM Detached Demo[tm] 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[tm] 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 its 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[tm] license can only be installed once on a particular system for any given combination of *product* and *version*. Detached Demo[tm] 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[tm] 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 Section 6.1.

---

rlm_detached_demox()

Remove RLM Detached Demo:sup`tm` 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[tm] 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[tm] 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.

---

rlm_detached_demox() will only remove a Detached Demo:sup:`tm` license created by the same version of RLM.

Back to Section 6.1.

---

rlm_diagnostics()

Print client-side diagnostics

```
RLM HANDLE rh;
char \*filename;

(void) 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 Section 6.1.

---

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()/rlm_act_request(). 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 For certain activation errors (rlm_act_request() or rlm_activate()) additional status will be contained in the returned license string. See rlm_activate() for more information.

Back to Section 6.1.

---

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 Section 6.1.

---

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 your 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 rlm_auto_hb(), rlm_set_auto_hb_isvdata() 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

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 Section 6.1.

---

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 (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 Section 6.1.

---

rlm_get_attr_lic_check()

Get license check information

```
#include "license.h"
RLM_LICENSE lic;
char \*license;
char \*hostid;
int status;

status = rlm_get_attr_lic_check(lic, &license, &hostid);
```

If you have enabled server license checking with *rlm_isv_cfg_enable_check_license()* AND you have disabled failures from this license check with *rlm_isv_cfg_no_server_license_fail()*, you can check the status of the returned license from the server with this call after you have attempted a checkout. This data will be valid after the checkout and before any other call that attempts to connect to the server.

The call will return pointers to the license and server hostid returned by the license server in the *license* and *hostid* parameters, if they are supplied as non-NULL pointers. You should **NOT** free or otherwise modify either the license or hostid strings returned from this call.

The return from the call will be one of the following values:

0	License from server verified correctly.
RLM_EH_BADPARAM	Either lic is NULL or the RLM_HANDLE associated with *lic* is NULL.
RLM_NO_SERVER_LIC	The server did not return a license.
RLM_LIC_BAD	The returned license did not verify correctly.
RLM_EH_LIC_WITH_NEW_KEYWORDS	The returned license has unknown keywords.

Back to Section 6.1.

---

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.

Note Prior to RLM v12.4, rlm_get_rehost() changed the working directory during the time of

the call. This caused problems with multi-threaded applications, specifically plugins to Adobe products. In RLM v12.4, the working directory is not changed during this call. There are no changes to the API or the file layout of rehostable hostids, so no changes to your product are needed.

Back to Section 6.1.

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 the hostid can be the string "invalid" in certain circumstances.

The value of *type* should be one of:

- RLM_HOSTID_32BIT
- RLM_HOSTID_DISKSN (Windows only)
- RLM_HOSTID_ETHER
- RLM_HOSTID_UUID (Windows only)
- RLM_HOSTID_USER
- RLM_HOSTID_HOST
- RLM_HOSTID_IP
- RLM_HOSTID_RLMID1
- 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 Section 6.1.

---

rlm_init(), rlm_init_disconn()

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);

handle = rlm_init_disconn(license, argv0, license_strings, promise);
```

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. If you plan to call *rlm_init_disconn()*, please read the section on Section 5.20 first.

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. List elements are separated by a semicolon on Windows and a colon on all other platforms. This parameter can also be a port@host specification. 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(). **If this string is dynamically allocated, it must remain valid for the lifetime of the RLM_HANDLE created by the rlm_init() call.**

For example:

```
5053@myhost
```

RLM Cloud

This can also be done for RLM Cloud access. In this case, set RLM_LICENSE to "port@host@-customer-name@password" (customer name MUST be all lowercase, even if it is mixed case on

the RLM Cloud server).

```
5053@ls1.rlmcloud.com@demo1@abc123
```

For more information see customer-license-file in the RLM Cloud manual.

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. 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. These licenses must be node-locked, uncounted (or SINGLE) licenses. Also, if this string is dynamically allocated, it must remain valid for the lifetime of the RLM_HANDLE created by the rlm_init() call.**

Note 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.

For example, to pass 2 licenses into RLM using the *rlm_init()* call, pass a string similar to the following as the third parameter to rlm_init() (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 ...>
```

If you are calling *rlm_init_disconn()*, **the fourth parameter** is how often you promise to contact the server. Specify *promise* in minutes. If you do not contact the server every *promise* minutes (by calling *rlm_get_attr_health()*), your licenses will be automatically checked back in by the license server, and the server will forget about you; subsequent calls to rlm_get_attr_health() will return RLM_EL_REMOVED (or alternately RLM_EL_NO_HEARTBEAT, RLM_EL_SERVER_DOWN, or RLM_EH_NOCLIENT).

Note 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 Section 6.2.

```
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.

- 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.

Client timeouts connecting to the license server

There are 2 different timeouts that clients can experience when talking to license servers: CONNECT timeouts and READ/WRITE communication timeouts.

By default, the CONNECT timeout is set to 10 seconds. To change this, set the environment variable RLM_CONNECT_TIMEOUT to a positive integer, e.g.:

```
% setenv RLM_CONNECT_TIMEOUT 5
```

to set the connect timeout to 5 seconds.

The read/write timeout is set to 5 seconds by default. To change this, set the environment variable RLM_COMM_TIMEOUT to 1000 times the timeout desired, e.g.:

```
% setenv RLM_COMM_TIMEOUT 10000
```

to set the comm timeout to 10 seconds.

Special note about MacOS

The RLM temporary directory on mac changed from /var/tmp to /Library/Application Support/Reprise in RLM v14.1. The main implication for this is that rehostable hostids should be revoked before upgrading to v14.1 and re-activated after. See appendix-j for more information.

One additional implication is that while this directory was writable in older macOS versions, it is no longer writable (at least on Catalina, and possibly on some earlier versions). So you will need to add a step to your installation procedures on macOS, to create /Library/Application Support/Reprise and set it to mode 777. If you do not do this, and the directory doesn't exist or is unwritable, you will get the new RLM_EH_NOTEMPDIR error from rlm_init(), and when RLM is started, it will log this message and exit:

```
% rlm

  The RLM temporary directory:
   /Library/Application Support/Reprise
   could not be created.

  Please create this directory with the following 2 commands:
   sudo mkdir "/Library/Application Support/Reprise"
  and
   sudo chmod 777 "/Library/Application Support/Reprise"
```

Also, in RLM v14.1 (and v14.1 only) the rlmsign utility will fail with some type of "unauthorized" error. This last error is corrected (i.e., a better error message) in later versions of RLM.

Back to Section 6.1.

---

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. The sum of the counts across all processes might be larger than the total available count due to sharing.

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 Section 6.2. 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.

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 Section 5.5.6).

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_cached()

Is the license cached on client node?

```
int cached = rlm_license_cached(license);
```

If this is a cached license, the return is 1, otherwise 0.

rlm_license_client_cache()

Returns the value of client_cache.

```
int cache_time = rlm_license_client_cache(license);
```

Returns the cache time for the license, in seconds.

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.

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_hold()

Retrieve license hold time.
rlm_license_host_based()

Is license host-based?

```

```

int host_based = rlm_license_host_based(license);
rlm_license_hostid()

Retrieve license hostid.

```
char \*hostid = rlm_license_hostid(license);
```

License servers can serve nodelocked licenses as well as floating licenses. If you want to know if the license was checked out from a license server, call *rlm_license_server()*. If you want to know if the license is nodelocked, call *rlm_license_hostid()* - if the license is floating instead of node-locked, this will return NULL. This call, however, returns the hostid from the LICENSE line, never the server's hostid. So for floating licenses, the return will be empty.
rlm_license_ismetered()

Is this license metered?

```
int ismetered = rlm_license_ismetered(license);
```

rlm_license_ismetered() returns non-zero for a metered license, - 0 otherwise. Note that 0 is returned if the handle passed in is invalid.
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_max_roam()

Retrieve maximum roam time.

```
int days = rlm_license_max_roam(license);
```

rlm_license_max_roam_count()

Retrieve maximum roam count.

```
int max_roam_count = rlm_license_max_roam_count(license);
```

rlm_license_max_share()

Retrieve maximum # of licenses that can be shared.

```
int max_share = rlm_license_max_share(license);
```

rlm_license_meter_counter()

Retrieve the meter counter for this license (if metered).

```
int counter = rlm_license_meter_counter(license);
```

rlm_license_min_checkout()

Retrieve license minimum checkout time.

```
int secs = rlm_license_min_checkout(license);
```

rlm_license_min_remove()

Retrieve license minimum remove time.

```
int secs = rlm_license_min_remove(license);
```

rlm_license_min_timeout()

Retrieve license minimum timeout.

```
int secs = rlm_license_min_timeout(license);
```

rlm_license_named_user_count()

Retrieve license Named User Count.

```
int count = rlm_license_named_user_count(license);
```

If count == 0, this is not a named user license.

rlm_license_named_user_min_hours()

Retrieve license Named User Minimum # of hours on the list.

```
int minimum_hours = rlm_license_named_user_min_hours(license);
```

The return from this call is only valid for licenses with named_user_count > 0.
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 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_roaming()

Retrieve "roaming" status of license.

```
int roaming = rlm_license_roaming(license);
```

If this is a roaming license, the return is 1, otherwise 0. Note that when you check the license out from the server requesting the roam, this is **not** a roaming license. The license is only roaming when the system is using the local roamed license without using the license server.
rlm_license_server()

Retrieve license server hostname.

```
char \*server = rlm_license_server(license);
```

If *license* is a license handle which has a valid checkout of a license which was granted from a license server, then the *rlm_license_server()* call will retrieve the hostname of the license server, otherwise an empty or NULL string will be returned. License servers can serve nodelocked licenses as well as floating licenses. If you want to know if the license is nodelocked, call *rlm_license_hostid()* - if the license is floating instead of nodelocked, this will return NULL.
rlm_license_share()

Retrieve license sharing spec.

```
int share = rlm_license_share(license);
```

share is a bitmap, with the bits defined in *license.h*:

- RLM_LA_SHARE_USER - Share if username matches.
- RLM_LA_SHARE_HOST - Share if hostname matches.
- RLM_LA_SHARE_ISV - Share if the isv-defined data matches.

All fields specified in the bitmap must match in order for the license to be shared.
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_soft_limit()

Retrieve license soft limit.

```
int soft_limit= rlm_license_soft_limit(license);
```

Note If licenses are pooled by the license server, the soft limit returned by this call is the sum of the soft limits of all pooled licenses. If some licenses do not have soft limits, the license count is used as the soft limit, thus this number could be equal to the license count.

---

rlm_license_start()

Retrieve license start date.

```
char \*start= rlm_license_start(license);
```

rlm_license_teams() - Retrieve RLM Teams URL

```
char \*url = rlm_license_teams(license);
```

If *license* is a license handle which has a valid checkout of a license which was granted by RLM Teams, then the *rlm_license_teams()* call will retrieve the URL of the RLM Teams server, otherwise an empty or NULL string will be returned.
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_user_based()

Is license user-based?

```
int user_based = rlm_license_user_based(license);
```

rlm_license_ver()

Retrieve license version.

```
char \*ver= rlm_license_ver(license);
```

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. Also note that if the request was satisfied by a TOKEN license on the server side, the version returned will be the greater of the requested version and the version of the first primary license which the TOKEN license utilizes.

Back to Section 6.1.

---

rlm_log(), rlm_dlog()

Log ISV-specific data.

```
#include "license.h"
RLM_HANDLE handle;
const char data[];
```

Log to report log:

```
(int) status = rlm_log(handle, data);
```

Any time after calling *rlm_init*, the *rlm_log()* function can be called to log data to the ISV server's report log (if it exists). The *rlm_log()* function will establish a connection to a license server if one does not already exist, then all subsequent *rlm_log()* and *rlm_dlog()* calls will operate with this server (until *rlm_close()* is called on the handle).

The data logged is a character string, which should not contain a newline character. The format of the data logged to the reportlog is:

```
log hh:mm:ss data from rlm_log call
```

Log to debug log:

```
(int) status = rlm_dlog(handle, (const char \*) data);
```

Any time after calling *rlm_init*, the *rlm_dlog()* function can be called to log data to the debug log. The rlm_dlog() function will establish a connection to a license server if one does not already exist, then all subsequent *rlm_log()* and *rlm_dlog()* calls will operate with this server (until *rlm_close()* is called on the handle).

The data logged is a character string, which should not contain a newline character. The format of the data logged to the debug log is:

```
mm/dd hh:mm (ISV name) ISV: data from rlm_log call
```

Return value:

rlm_log() and *rlm_dlog()* return 0 for success, and on failures:

- RLM_EH_NOHANDLE - Called without a valid RLM_HANDLE.
- RLM_EH_NO_REPORTLOG - rlm_log() called but the server does not have a reportlog.

Note The maximum length of a logged string is RLM_MAX_LOG (currently 256). However, logging strings much longer than 60 bytes will generally create wrapped lines in the debug and report log files.

Back to Section 6.1.

---

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, and it also goes to every server in the list and retrieves from each server a list of available licenses.

---

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 Detached Demo™ 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.

---

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 *rlm_products()* returns the list of valid roamed products on the local node, **whether or not it can check out an rlm_roam license**.

---

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() (not recommended), 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.

Function	Description
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.
char ***rlm_product_akey(products)**	Returns the akey= attribute.
int **rlm_product_client_cache()**	Returns the value of the client_cache parameter.
char ***rlm_product_customer(products)**	Returns the customer attribute.
char ***rlm_product_contract(products)**	Returns the contract attribute.
int **rlm_product_count(products)**	Returns the license count.
int **rlm_product_current_inuse(products)**	Returns the license count in use.
int **rlm_product_current_resuse(products)**	Returns the # of reservations in use.
int **rlm_product_hbased(products)**	Returns the HOST-BASED count.
int **rlm_product_hold(products)**	Returns the license hold time.
char ***rlm_product_hostid(products)**	Returns the license nodelock hostid, if it exists.
int **rlm_product_isalias(products)**	Returns non-zero for an alias license, 0 otherwise.
int **rlm_product_isfloating(products)**	Returns non-zero for a floating license, 0 otherwise.
int **rlm_product_ismetered(products)**	Returns non-zero for a metered license, 0 otherwise.
int **rlm_product_isnodelocked(products)**	Returns non-zero for a nodelocked license.
int **rlm_product_issingle(products)**	Returns non-zero for a single license, 0 otherwise.
char ***rlm_product_issuer(products)**	Returns the issuer attribute.
int **rlm_product_max_roam(products)**	Returns the maximum roam time.
int **rlm_product_max_roam_count(products)**	Returns the max # of these licenses which can be roamed.
int **rlm_product_max_share(products)**	Returns the max number of processes that can share this license.
int **rlm_product_meter_counter(products)**	Returns the meter counter for this product (0 if not metered).
int **rlm_product_meter_cur_count(products)**	Returns the meter count for this product (0 if not metered).
int **rlm_product_min_remove(products)**	Returns the minimum rlmremove time.
int **rlm_product_min_checkout(products)**	Returns the license minimum checkout time.
int **rlm_product_min_timeout(products)**	Returns the minimum timeout time.
int **rlm_product_named_user_count(products)**	Returns the named user count.
int **rlm_product_nres(products)**	Returns the # of license reservations.
int **rlm_product_num_roam_allowed(products)**	Returns the # of roaming licenses allowed.
char ***rlm_product_options(products)**	Returns the product options.

int **rlm_product_personal(products)**	Returns the personal license reservation count.
int **rlm_product_roaming(products)**	Returns the # of licenses currently roaming (for roaming licenses).
char ***rlm_product_server(products)**	Returns the license server's hostname (from the server's license file).
char ***rlm_product_start(products)**	Returns the product's start date, if present.
int **rlm_product_share(products)**	Returns the license share flags. (Share flags (RLM_LA_SHARE_xxx) are defined in *license.h*)
int **rlm_product_soft_limit(products)**	Returns the license soft limit.
int **rlm_product_thisroam(products)**	Returns 1 if this license is a roaming license.
int **rlm_product_timeout(products)**	Returns the current license timeout.
int **rlm_product_tz(products)**	Returns the license timezone specification.
int **rlm_product_tokens(products)**	If 0, this is a normal license. If non-zero, this is a token-based license.
int **rlm_product_type(products)**	Returns the license type (TYPE= parameter). (License type flags (RLM_LA__xxx_TYPE) are defined in *license.h*)
int **rlm_product_ubased(products)**	Returns the USER_BASED count.

Note All functions with char * returns can return NULL for cases where the data does not exist, so you must check for a NULL return. For example, rlm_product_server() called on an unserved license will return NULL.

Note client-cached licenses on the client side will never be returned by *rlm_products()*. The rlm_product_client_cache() call returns the value of the cache time (in seconds) from the server's license.

Certain of these items will always be 0 for node-locked, uncounted licenses. These are:

- current_inuse
- current_resuse
- min_remove
- nres
- num_roam_allowed
- timeout

Note The list of products may contain products that cannot be checked out at any given time, if all the licenses are in use, the maximum # of roaming licenses has been reached, etc. 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 if a hostid is specified.

Also note that the following licenses will never be returned by rlm_products():

- Detached Demo[tm] licenses
- reserved license names (e.g., *rlm_server*, *rlm_server_enable_vm*, *rlm_no_server_lock*)
- licenses passed in the 3rd parameter to rlm_init()
- Client cached licenses

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 Section 6.1.

---

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 into 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 Section 6.1.

---

rlm_set_active()

Inform RLM of activity status of application.

```
#include "license.h"
RLM_HANDLE rh;
int active;

(void) rlm_set_active(rh, active);
```

rlm_set_active() informs RLM that the application is *active* (active=1), or inactive (active=0). This is used with *rlm_auto_hb()* in order to allow your application's license to be timed-out with the license administration TIMEOUT option. The activity state applies to all checked-out licenses on the handle.

If you call *rlm_auto_hb()*, heartbeats will be sent to the server whenever your application is running, whether you are actively processing or not. This means that an application which is simply waiting for user input will continue to send heartbeats, and will appear active to RLM. If you would like your license administrators to be able to time-out the licenses from applications which are in such an idle state, use the *rlm_set_active()* call to inform RLM when your application is active or inactive.

Note If you use manual heartbeats the rlm_set_active() call is not necessary, since you would only send heartbeat requests by calling rlm_get_attr_health() calls when the application is active.

A call to *rlm_checkout()* will set the active flag to 1 (ie, the application is active) whether the checkout succeeds or not.

Back to Section 6.1.

---

rlm_set_attr_keep_conn()

Set "keep connection" status for RLM.

```
#include "license.h"
RLM_HANDLE rh;
int conn_status;

rlm_set_attr_keep_conn(handle, conn_status);
```

When multiple checkouts are performed in a single application, RLM automatically create a new connection to a license server if the current license server cannot provide the license required. This operation is transparent to the application, and the RLM software maintains a list of connected servers which can be queried (in the order connected) for any new license request.

When a checkout request fails, RLM can either close down the connection to the license server or keep it open. If your product checks out multiple licenses, an optimization is to keep these connections open (at the expense of the TCP/IP overhead for the connection). The default is to close the connection, which will be correct for the vast majority of applications. However, if you would like to keep the connection open, you can call:

```
rlm_set_attr_keep_conn(handle, 1);
```

Please note that making this call will not affect the functioning of RLM in your application - it is an optimization only.

Back to Section 6.1.

---

rlm_set_attr_logging()

Turn server logging on or off.

```
#include "license.h"
RLM_HANDLE rh;
int log;

rlm_set_attr_logging(handle, log);
```

By default, RLM servers log all license checkout and checkin activity.

You can call *rlm_set_attr_logging()* any time before a checkout or checkin request, and if the value of *log* is non-zero, the server will log the the checkout/checkin status in both the debug and report log files. If the value of *log* is zero, the server will not log anything about the checkout/checkin.

This is useful, for example, if you want to check out a product which should never succeed, in order to see if your license server has been compromised. The resulting checkout activity will not appear in any server log and will not confuse your customers. You should remember to turn logging back on after doing this, otherwise your license server logs will contain no checkout/checkin activity.

Back to Section 6.1.

---

rlm_set_attr_password()

Set license password-string for future operations.

```
#include "license.h"
RLM_HANDLE rh;
char \*password_string;
```

```
rlm_set_attr_password(handle, password_string);
```

Each license can contain a password. If the license contains a password, only processes which specify the matching password-string, will be able to use (either check out, or view) the license.

You can call *rlm_set_attr_password()* any time before a checkout or status request, and the value of the password-string can be changed for subsequent requests.

Note The setting of the environment variable RLM_LICENSE_PASSWORD will be the default, but this call will override the value set in RLM_LICENSE_PASSWORD.

The intended use of this capability is to allow a single license server to serve licenses for several independent customers. Each customer would be given a password-string, which would enable access to all their licenses, but not the licenses of other customers.

If a license does not specify a password, calling *rlm_set_attr_password()* (or setting RLM_LICENSE_PASSWORD) will have no effect.

Back to Section 6.1.

---

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);
```

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 RLM selects a reference hostid automatically, and you should never need to make this call.

Back to Section 6.1.

---

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);
```

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. 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 Section 6.1.

---

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()*.

You can call *rlm_set_environ()* any time (even after the first *rlm_checkout()* call), and the new user, host, and ISV-defined parameters will apply to all subsequent checkouts, until you call *rlm_set_environ()* again. The original settings will continue to apply to *rlm_products()* calls, however.

Note RLM always treats usernames and hostnames as case-insensitive.

Back to Section 6.1.

---

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.
<4 or >6	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.

Note Do not call *rlm_sign_license()* in an application or utility that ships to customers. Doing so will cause your private key to be included in the application executable or binary, which could expose it to hackers, possibly enabling them to create counterfeit licenses for your product.

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)
                -errorelse
        {
          (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 Section 6.1.

---

rlm_skip_isv_down()

Enable "skip" of license servers where your ISV server isn't running.

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

(void) rlm_skip_isv_down(rh);
```

RLM keeps track of which license servers in the list have your ISV server running on them.

If either:

- The client cannot connect to the server, or
- An attempt is made to check out a license from a server and the rlm server returns a status indicating that this ISV is not present,

then the server is flagged as not having your ISV server.

If you call *rlm_skip_isv_down()*, then future *rlm_checkout()* and *rlm_products()* calls will not attempt to use this license server.

Note By default, RLM will attempt all operations on all servers.

You can call *rlm_skip_isv_down()* any time after calling *rlm_init()*.

If you would like to give your user the opportunity to attempt to use these servers again, call:

```
(void) rlm_forget_isv_down((RLM_HANDLE) rh)
```

rlm_forget_isv_down() will cause RLM to attempt to use all license servers again, until it determines that your ISV server is not running.

Note RLM will attempt a connection one time to servers that don't have your ISV line in them. If the server is not up or if your ISV server isn't present, then that server won't be checked again.

Back to Section 6.1.

---

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. The status returned by *rlm_stat()* is only meaningful if called after *rlm_init()* or *rlm_init_disconn()* and **before any other RLM function**.

Calls to rlm_stat() after other RLM function calls return undefined results.

For a list of status returns, see Section 6.2.

6.2 Appendix B – RLM Status Values

The RLM API functions return status (via the rlm_stat() and rlm_license_stat() (see rlm_license_XXXX()) calls.

6.2.1 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_DEMOEXISTS	-137	Demo already created on this system.
RLM_EH_DEMOWRITEERR	-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	Semaphore 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 (RLM Cloud 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 RLM Cloud 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).

6.2.2 In addition, rlm_activate()/rlm_act_request() will return the following errors:

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_FULFILL	-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 (ActPro).
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).
RLM_ACT_DB_READERR	-1026	Activation database read error (ActPro).
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	RLM Cloud: 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	RLM Cloud: Can't write licf table.
RCC_PORTAL_CANT_WRITE_FULFILL	-1041	RLM Cloud: 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.

6.2.3 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 for product	rlm_checkout() did not find a product to satisfy 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 initialization, 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 queuing 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	This license checkout causes the license usage to license soft_limit 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 your 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	Metered licenses only work with with a license require server 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_EL_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	RLM_ROAM_EXTEND format is 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 RLM Cloud.
RLM_EL_BAD_JSON	-69	Bad JSON in returned message	This is an internal error in web services processing with RLM Cloud.
RLM_EL_BADHANDSHAKE	-70	Bad handshake on web services checkout	This is an internal error in web services processing with RLM Cloud.
RLM_EL_HELPER_ERR	-71	rlm_helper error	This is an internal error in web services processing with RLM Cloud.
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.

6.3 Appendix C - RLM Hostids

6.3.1 RLM Host IDs

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

Table 6.1. RLM's host identification (hostid) types are:

Type	Meaning	Example	Notes
ANY	runs anywhere	hostid=ANY	
DEMO	runs anywhere for a demo license	hostid=DEMO	
serial number	runs anywhere	hostid=sn=123-456-789	Used to identify a license, any string up to 64 characters long.
disksn	Hard Disk hardware serial number	hostid=disksn=WDWX60AC94686O	Windows only.
32	32-bit hostid, native on Unix, non X86 based platforms	hostid=10ac0307	This is the volume serial number on windows, and is not recommended.
ip (or internet)	TCP/IP address	hostid=ip=192.156.1.3	Always printed as "ip=". Wildcards allowed.
ether	Ethernet MAC address	hostid=ether=00801935f2b5	Always printed without leading "ether=".
rlmid1	External key or dongle	hostid=rlmid1=9a763f21	External key or dongle.
uuid	BIOS uuid	hostid=uuid=699A4D56-5BE1-DB3C-27A8BEB8011A	Windows only.
user	User name	hostid=USER=joe	Case-insensitive
host	Host name	hostid=host=melody	
gc	*Google Compute Engine*	*hostid=gc=3797742226458986650-.k6qt9v5h38w2adwqgc9fdhdf3w0m761p*	*Linux only, DEPRECATED.*

Warning The Google Compute host ID is deprecated as of v14.2 and should not be used.

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
```

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 * 1 or 2 Disk serial numbers (windows only) * up to 3 IP address * up to 5 ethernet MAC addresses (ether=) * up to 6 RLMID portable hostids * a UUID hostid, if present * a minimum of 3 ISV-defined hostids (usually more, but guaranteed to be at least 3)

Note In RLM v15.0, the server will add the client's external IP address to the list of IP addresses supplied by the client. This IP address is not added if it is the same as one of the client-supplied IP addresses.

6.3.2 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):

Listing 6.1. From the shell

```
% setenv RLM_LINUX_ETHERNET_ITERATIONS 20000
```

or

Listing 6.2. In your application

```
rlm_putenv("RLM_LINUX_ETHERNET_ITERATIONS=20000")
```

6.3.3 Windows hostids

Ethernet

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.

disksn

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".

6.3.4 Miscellaneous Notes

RLMID

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 Section 6.4.

Wildcards

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

A wildcard may be used in the host type hostid, for example:

```
"hostid=host=*.stanford.edu" or "hostid=host=*.reprisesoftware.com"
```

6.3.5 Disabling standard RLM Hostids

You can disable certain hostid types in your application in rlm_isv_config.c.

Note 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_DISKSN
- RLM_DISABLE_H_IPV6
- RLM_DISABLE_H_UUID

6.3.6 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 changeable, 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.
disksn	std	
gc	std	Deprecated.
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

6.4 Appendix D – 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 SafeNet, Inc.

6.4.1 Installing RLMid1 Devices on Windows

Installation on a target system can be accomplished in two ways:

1. Windows automatic installation, OR
2. Use the RLMID1 driver installer (from the Reprise Software website) to do the driver installation.

Plug the USB device into your computer. Windows will search for and install the drivers automatically.

If for some reason Windows fails to update the driver automatically (usually because the target system is not connected to the internet), use the driver installer located at:

RLMid1.zip

Note This download requires login from the Reprise website.

The driver can also be found on the SafeNet site.

Unzip the installer and run it on the target system.

Note 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.

6.4.2 Installing RLMid1 Devices on Linux

To install the necessary drivers for RLMid1 devices on Linux, you will need the Sentinel HASP/LDK Runtime Installer for Linux. At the time of writing this, instructions can be found here:

https://docs.sentinel.thalesgroup.com/ldk/LDKdocs/Install/Installation%20Guide/RTE_for_Linux/Install.ht

The runtime installer sets up a daemon that is used to access the hardware key.

6.5 Appendix F - 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.

6.5.1 For ISVs:

https://www.reprisesoftware.com/support/publishers/faq/

6.5.2 For License Administrators:

https://www.reprisesoftware.com/support/admin/faq/

6.6 Appendix G - RLM version history

This section describes the various versions of RLM, and the features added in each version.

Table 6.2. V16.0 – May, 2024

Features Added:
New web management interface for the on-premises license server.
HTTPS now enabled by default.
Self-signed certificates now automatically generated.
Diagnostics now viewable directly from web interface.
New user roles/permission levels.

Table 6.3. V15.2 – December, 2023

Features Added:
Warning no longer presented to user when C:\ProgramData\Reprise\isvname\ doesn't exist.
License path no longer includes temporary directory if the temporary directory does not exist.
rlm_stat() no longer returns an error if the C:\ProgramData\Reprise\isvname folder is missing.
Temporary licenses are now correctly created with version number equal to "parent" license.
Using unicode in company names no longer causes issues with portal users on RLM Cloud.
Server usage on the RLM Cloud customer portal now loads additional pages correctly.
log.php no longer fails to downloads report logs.
ISVs are now able to disable certificate revocation in RLM Cloud.

Table 6.4. V15.1 – April, 2023

Features Added:
Added support for ISV_LICENSE (uppercase) in addition to isv_LICENSE in casesensitive file systems.
License checkout now obeys single license count on Linux when using multiple checkouts in single process.
RLM embedded web server changed from GoAhead to Mongoose.
RLM web server can now be configured to access via HTTPS.
RLM web server now requires login to access.
Removed restriction of running RLM as root/administrator.
Removed option to edit any server files using 'Edit License File'.

Table 6.5. V15.0 – May, 2022

Features Added:
The server now adds the client's external IP address to the list of hostids.

Diagnostics only appears in the RLM web interface when edit_options, edit_rlm_options, logfiles, or all privs are specified for the user.
RLM_LINUX_ETHERNET_ITERATIONS to limit ethernet device scan iterations.
EXEMPT_MAX and EXEMPTALL_MAX added.
Activation Pro Features:
PID added to debuglog to sort out multipel _mklic invocations.
Prepended text can now be 30kb.

Table 6.6. V14.2 – Mar, 2021

Features Added:
rlmstat lists server license pool# and ID (if specified).
RLM_LICENSE/ISV_LICENSE strings leading/trailing whitespace trimmed.
Personal licenses no longer take a count.
You can lock out 2 old ISV server names.
rlm -verify will cause the servers to verify their licenses, then exit.
RLM exits after 10 minutes if users have blank passwords (-z to disable).
Docker detection (non-Windows).
RLM Teams support. RLMTEAMS license line.
RLM web interface no longer stores username/permissions in cookie.
gc= (Google compute) hostid no longer supported.
Activation Pro Features
PHP v8 supported.
Expiring license and activation key data displayed in dashboard.
"Create license" capability added to GUI.

Table 6.7. V14.1 – July, 2020

Features Added:
Temporary Licenses supported with RLM Cloud.
Personal Licenses.
Detailed reportlog now has client IP address in OUT records.
You can now lock out old versions of your license server (old ISV names).
Startup tolerance for ASH licenses.
exptime= keyword added.
License type now accepts "subscription".
rlm_act_rehost_revoke() call .
rlm_product_isalias, rlm_product_isfloating(), rlm_product_isnodelocked(), rlm_product_single() calls added.
Activation Pro Features:
HTTPS support for Activation Pro license generator.
Debug log can be displayed in chronological or reverse chronological order.
Debuglog no longer logs all entries in the blacklist.
RLM_ACT_DIAGNOSTICS to create activation diagnostics to a file.
New disabled activation keys report.
You can now upload your logo to replace the Reprise Logo in the GUI (on *.hostedactivation.com) the debuglog table will be purged every night of all data > 14 days old.
admin->delete revoked allows deletion of revoked permanent licenses.
Product definition table includes a displayorder column (to control product choicelist).
The user_def column in the keyd table is now 32 chars. (Unused by ActPro).
The login inactivity timeout is now 60 minutes.

Table 6.8. V14.0 – November, 2019

Features Added:
HTTPS support for RLM Cloud.
rlm_isv_cfg_disable_hostids()
client_cache, hold, host_based, max_roam, min_checkout, min_timeout, nmamed_user, soft_limit, and user_based can all now be set to 0 in a license to be signed.
ASH licenses support "6hour" check type.
RLM Web services API now supports reservation type/strings/seconds and username/password.
RLM web interface license usage now also displays license pool status.
Support for rlm_products() in HTTPS (beginning in BL3).
Activation Pro features:
Fulfillments tab displays activation host name.
Fulfillment tab allows selection by notes field.
Product definitions now have a notes field.
Activation keys can be cloned.
GUI no longer accepts relative license expiration date for subscription type keys.
Customer browser displays activation keys as well as customer info.
Timezone can be set on hostedactivation.com.
Support for "6hour" ASH check.
rlm_activate() allows specification of company/contact info.

Table 6.9. V13.0 – February, 2019

Features Added:
UUID support on 64-bit Mac.
rlmstat -f option for extended information.
License server authentication of clients.
rlm_product_named_user_count() call.
Hostname added to web server activation confirmation page.
rlm_product_meter_counter(), rlm_license_meter_counter() calls.
rlm_product_start() call.
"Activate License" in web interface controlled by edit_options privilege.
Activation Pro features:
Reseller support.
Customer portal now part of standard URL – no configuration required.
Customer portal accounts can be created in the "add customer" form.
Option to blank the remote_host parameter (for GDPR compliance).
Admin "check database" command creates blank entries in the keyf table so the.
"Unfulfilled Keys" report works correctly.
Product definition/activation key forms include license start date.

Table 6.10. V12.4 - July, 2018

Features Added:
Alias licenses added.
token_locked depricated, token_bound added.
rlm_license_ismetered(), rlm_product_ismetered() calls added.
Check server license can return status rather than failing immediately.
rlm_auto_hb() has new parameter.

rlm_product_meter_cur_count() call added.
RLM web interface disables options and license file editing functions if logins are not enabled.
Activation Pro Features:
Database queries now prevent SQL injection.
User-defined reports added with auto-email capability.
New “edit key” user type added.
actpro_keyvalid() web services call added.
rlm_act_keyinfo2() added.
Sorting works correctly on first fulfill and last check fields in fulfillments.
Product/activation key screens add misc parameters (several missing in 12.3).

Table 6.11. V12.3 - Oct, 2017

Features Added:
rlm_failover_server_activate failover mode
RLM report log logs the timezone
Roamed licenses can have their roam time extended when disconnected
Failover servers display a list of valid primary server licenses.
Control over the rehostable hostid checks
Set the rehostable hostid reference hostid (rlm_set_attr_reference_hostid())
Activation Pro Features:
Rlmact.mysql renamed to actpro.php
Optional parameters are broken out in the GUI
Duplicate a product definition to create a new one
Activation key browser includes the contact email
Custom fields for a company can be specified as numeric to sort properly
On login, an invalid username or password displays “login incorrect”
Newlines are replaced with spaces in the license field in reports

Table 6.12. V12.2 - Feb, 2017

Features Added:
Dynamic Reservations.
rlm_product_server() call.
spec in rlm_failover_server license can now omit port-number.
Illegal character restriction in customer= field relaxed.
UUID hosids (Windows).
rlmstat now accepts the “-I” switch, to display ISV-defined data.
The report log INUSE records now include the share handle for shared licenses.
rlm_get_attr_lfpath() returns char * rather than const char *.
rlmstat -I switch (to display ISV-defined data).
Activation Pro features:
rlm_act_fulfill_info() call.

Table 6.13. V12.1 - June, 2016

Features Added:
-l switch to RLM.
License transfers use hostname-RLM-Transfer as username and isv string.
RLM Comm protocol optimized.
Server Time Jump reportlog message.

Whitespace removed from user and host names.
rlm_activate()/rlm_act_request() copy url and akey parameters.
PURGE_REPORTLOG added.
Activation Pro Features:
Merge company/contact.

Table 6.14. V12.0 - December, 2015

Features Added:
rlm_product_hostid() call added.
Shared license can now be metered.
The install Windows service menu item in the RLM web interface is removed.
Licenses sorted by _id
Servers log expired licenses on startup.
Alternate Nodelock Hostids
Install service command takes -user and -password
VS2015 support
rlm_product_hostid()
rlm_set_req_opt()
rlm_act_keyinfo()
SSL entry points renamed on most Unix platforms.
Activation Pro Features:
user_def field added to keyd table.
Product choicelists now sorted alphabetically.
Search added to debug log.
Unencrypted requests can now be disabled.
Normalized date formats.
Admin function to delete keys with revoked rehostable fulfillments.
Arbitrary text can be added to the license via product definition.
rlm_act_keyinfo()

Table 6.15. V11.3 - April, 2015

Features Added:
RLM rejects ethernet devices named “dummy*” on Linux.
report log DENY records now include the process ID.
In rlmstat and the RLM web interface, the field previously labeled “transactions” is now labeled “checkouts”.
The contents of the ISV-defined string can now be used as a hostid.
rlm_diagnostics() call.
Activation Pro Features:
New improved and expanded ActPro Web Services interface.
For product definitions, the default for “create issued=today?” changed to “yes”.
Contacts now have an associated “Contact Type”.
The “About” tab now lists the activation pro license expiration date.
Users with “portal” access are no longer allowed to log in on the main actpro site.
A product can now be marked “obsolete”, in addition to “inactive”.
Default values can be set for the product editing and activation key editing screens.
User accounts can now be edited, from the Admin->Users screen.
An administrator can set the list of visible tabs for any user.

The “Alternate Server Hostid” choice is added for products and activation keys.

Table 6.16. V11.2 - November, 2014

Features Added:
Alternate Server Hostids.
rlm_license_count() call added.
The RLM version restrictions on client-side roam files have been relaxed to allow the “oldest compatible version”. As of 11.2, the oldest compatible version is 11.0.
Cached licenses no longer disable roaming.
Activation Pro Features:
Support for Alternate Server Hostids.
Audit trail for created/edited/deleted rows in the product, activation key, contact, company, and blacklist tables.
The date of creation of product definitions, activation keys, and customer contacts is now displayed in the appropriate browser.
Product Definitions and Activation Keys can now specify the allowed hostid types.
The “include activation key in license” option now allows you to generate licenses without akey, with akey, or both.
When the actpro server returns a RLM_ACT_BAD_HOSTID_TYPE status, it now returns a decimal integer indicating which hostids are valid.
The licf table now records the lst time rlm_act_keyvalid() was called.
The product, activation key, and fulfillment browsers now allow deletion of multiple items.

Table 6.17. V11.1 - June, 2014

Features Added:
RLMid1 dongles now supported on linux (x86_l2, x64_l1).
The first parameter of rlm_init() now takes a list of license files.
Setting RLM_ROAM=today causes a roam to end today at midnight.
Licenses now have an optional _id= parameter to identify for options file use.
For metered licenses, the user can now decrement the counter in the RLM web interface.
RLM now supports Google Compute Engine hostids (gc=).
Activation Pro features:
Manual rehostable hostid creation and revocation are now supported, for machines with no internet connection.
New rlm_act_keyvalid_license() call returns the license for a given activation key.
Expired rehostable licenses can now be revoked.
New rlm_get_rehost() call retrieves the rehostable hostid for a given product name.
rlm_act_revoke_reference() allows revocation of a rehostable on machines where the hostid has disappeared or gone bad.
Activation Pro logs the expiration date of the license and displays in the fulfillment screen.
The Customer Portal is set up after installation, in the Admin/Portal tab.
Normal-Regen activation type added.

Table 6.18. V11.0 - Feb, 2014

Features Added:
Ipv6 support (x86_l2, x64_l1, x86_w3 and x64_w3 only).
Auto proxy detection on Windows.

Applications can refuse to run with generic server.
(rlm_isv_cfg_disable_generic_server())
akey= license attribute.
rlm_products_akey() and rlm_license_akey() calls added.
rlm_license_uncounted(), rlm_license_single() calls added.
Client caching turned off on HP systems.
Activation Pro Features:
New tab-based UI.
New Customer portal.
Product Definitions allow multiple licenses.
Auto-generate akey= license attribute.
Activation Keys can override license version.
rehosts controls the # of times a rehostable hostid can be revoked.
Server writes debug logging to database.
Bulk-load of customer data.
Customization of the customer table.
rlm_isv_cfg_actpro_allowed_hostids() call in rlm_isv_config.c to control allowed.
Activation Pro hostid types.
rlm_act_keyvalid() call.
rlm_act_info() now returns RLM_ACT_KEY_DISABLED if appropriate.

Table 6.19. V10.1 - July, 2013

Features Added:
client_cache license option added.
CLIENT_CACHE license admin option added.
Server logs unreadable license files.
rlm_auth_check() call.
rlm_isv_cfg_enable_check_license() added.
License now available as search criteria in ActPro fulfilled licenses report.

Table 6.20. V10.0 - Jan, 2013

Features Added:
Disconected server-server license transfer.
rlm_init_disconn() for disconnected operations.
RLM will now broadcast to find a server on the local network.
Roaming is disabled if the license server uses a transient hostid.
The public/private keys are now declared as “const unsigned char”.
You can disable disk Serial Numbers which require admin rights.
You can disable RLM’s clock windback checking.
Option to make roamed licenses SINGLE rather than UNCOUNTED.
RLM web interface now supports user login, with access rights.
RLM web inteface only displays commands which the user can execute.
RLM web interface doesn’t display “Manage Windows Service” on non-Windows systems.
If RLM processes multiple license files, it will attempt to find a good ISV server path.
Browsers connecting on RLM’s main port are redirected to the webserver port.
Report log logs all licenses in use both at start and at the end.
Roamed license time extension logged in report log (and debug log).
RLM web interface allows editing license files.

RLM checks that the debug log is writable when installing service.
rlm_product_customer(), rlm_product_contract(), rlm_product_issuer(), rlm_product_exp_days() API calls added.
rlm_checkout_product() call.
rlm_act_info() call.
ISV_mklic not built by default.
RLM web interface allows the Activate License command customization.
INTERNET_GROUP option.
actpro_demo.o removed, actpro_demo.c added w/build rules (v10.0BL3).

Table 6.21. v9.4 - July, 2012

Features Added:
ISVNAME_ACT_URL overrides url in rlm_activate()/rlm_act_request() call.
actpro_demo added to RLM kit, removed from actpro kit.
Hostname hostid types now accept wildcards.
RLM utilities now accept the -z password option.
When installing RLM as a service on Windows, the installation now starts and stops the service to trigger firewall prompts.
The hostid list for rlm_activate() has been expanded to RLM_ACT_MAX_HOSTID_LIST characters (205) - including the "list:" prefix.
rlm_activate()/rlm_act_request() can now process replies from firewall and anti-virus software which splits a single message into 2 messages.
Clock windback detection checks internet time servers (if reachable) (removed sometime around RLM v10).
RLM servers write heartbeat messages back to the client (for v9.3 and newer clients) in order to detect systems which have been shut down.

Table 6.22. v9.3 – February, 2012

Features Added:
License rehosting via refresh activation and rlmrefresh utility is DEPRECATED.
License rehosting added with the new rehost hostid type and rlm_act_revoke() (requires Activation Pro).
x86_l1 platform obsoleted.
rlmsign now checks the validity of keywords in keyword=value pairs.
Metered licenses added.
rlm_activate() call added to replace rlm_act_request().
Activation can now create UPGRADE licenses for single and nodelocked, uncounted license types.
Client-side diagnostics now list all embedded string licenses in addition to other node-locked licenses.
Server-side diagnostics now output the RLM and ISV server option file info.
RLM now enumerates the ethernet devices on linux rather than using eth0-7.

Table 6.23. v9.2 - September, 2011

Features Added:
disksn hostid (disk hardware serial number) added on Windows.
License Passwords can now be specified on the ISV line.
rlc (internet activation) verifies the format of date input.
rlm_license_line_item() call added.
Product Definition names can be edited in rlc (Internet Activation).
Fulfillment count for refresh activations limited to 1.

"_primary_server" keyword added for rlm_failover licenses.
Generic rlmrefresh utility removed; replaced with ISV-specific utility.
Many changes to Activation Pro.

Table 6.24. v9.1 - May, 2011

Features Added:
RLM Activation Pro introduced.
Passwords on individual LICENSE lines.
rlm_set_attr_password() call.
disable=TerminalServerAllowRD attribute.
rlm_set_attr_logging() call.
rlm_add_isv_hostid() API changes (added parameter).
rlm_set_environ() call can be made any time before a checkout.
RLM_ACT_NO_ENCRYPT to prevent rlm_act_request() encryption.
rlm_auto_hb() period now set to min of 2 seconds.
rlm_errstring() now formats all errors including activation errors.

Table 6.25. v9.0 - December, 2010

Features Added:
GUI license generator.
Support for license server farms.
rlmstat reports on expiration dates.
New license checkout debugging capability/utility.
rlmsign takes the optional -maxlen parameter.
New rlm_failover_server license.
New rlm_no_server_lock license.
New keyword=value parameters on ISV line.
rlm_errstring_num() API call.
rlm_license_detached_demo() API call.
Multiple GROUP lines now concatenate in OPTIONS files.
New license administration REMOVE privilige.
Activation supports SINGLE and UPGRADE licenses.
rlm_act_request() allows white space in activation key.
rlm_act_request accepts hostid lists.
rlm_act_requests accepts a count of 0 to fulfill all remaining floating licenses.
rlm_act_request() encrypts traffic to the activation server.
rlm_act_reqeust() rejects newlines in the "akey" and "extra" parameters.

Table 6.26. v8.0 - Jan, 2010

Features Added:
Optimized license sharing.
Client and server side diagnostics to aid solving problems in the field.
When running as a service, RLM changes working directory to binary directory.
RLM logs the client machine's OS to the report log.
RLM logs the client's argv[0] to the report log.
Eval directory removed from Unix and Mac kits.
RLM web interface shows all license file and log file paths.
RLM web interface puts all activated license files into the directory specified with -c.
Single-quote and back-quote characters are now legal in license and option files.

max_roam_count license keyword.
Custom hostid comparision routine for ISV-defined hostids.
rlm_putenv() call for Windows DLLs.
Enforcement of legal ISV-defined hostid strings.
Activation: rlc adds ISV-defined notes to activation keys.
Activation: rlc checks for bad or missing hostids.

Table 6.27. v7.0 - June, 2009

Features Added:
Server-Server license transfers.
License key email from Reprise now contains ISV name.
Ability to return a roamed license unconditionally.
rlm_isv_config() allows you to disable older license servers.
Unix makefile target for no-Openssl version of library.
rlm_server_enable_vm license for enabling servers on virtual machines.
rlmanon is on kit (missing prior to v7).
rlmhostid no longer reports 32-bit hostids on linux, mac, or netbsd systems.
rlmhostid no longer prints "known bad" 32-bit hostids (0, 0xffffffff, 0x7f0100).
RLM_EL_FAILED_BACK_UP status when failed server restarts.
Failover servers no longer pool licenses from failed servers.
rlm_auto_hb() attempts to re-acquire the same license which was lost.
License line checksum (_ck=).
rlmid3 machine fingerprinting option removed.
New status parameter to notification handler for rlm_auto_hb().
rlm_license_user_based() API call.
rlm_license_min_remove() API call.
rlm_license_host_based() API call.
reactivate and refresh activation types.
License rehosting via refresh activation and rlmrefresh utility.
RLC allows you to specify the activation key.
RLC allows specification of additional license parameters in the activation key.
Whitelist and blacklist in activation.
Activation keys have an expiration date.
Activation allows RLMID devices.
Activation uses isv-defined, RLMID, ethernet, and 32-bit devices, in that order.
Activation no longer accepts "known bad" 32-bit hostids (0, 0xffffffff, 0x7f0100).

Table 6.28. v6.0 - January, 2009

Features Added:
Platform-independent ISV server settings and the Generic ISV server.
RLC refresh button.
RLC allows fixed expiration dates.
rlmid2 hardware key.
rlmid3 machine fingerprinting option.
rlm_act_request() processes HTTP redirects.
rlm_act_request() supports HTTP proxy servers.
ISV servers increase their open file limit.
ISV lockfile in C:\rlm removed.
port@host can be specified as host@port.

RLM_LICENSE environment and the -c option can contain directories.
RLM default port # changed from 28000 to 5053.
RLM admin port # changed from 9000 to 5054.
UPGRADE licenses.
min_checkout.
rlm_detached_demo() API call.

Table 6.29. v5.0 - May, 2008

Features Added:
Serial Number hostid type.
rlmID1 hardware key.
hostid lists.
ISV servers don't exit on reread if no license file exists.
RLM activation forces client clock to be within 7 days of activation server.
GUI license generator.
rlm_act_admin renamed to rlc.
Virtual machine detection in ISV servers.
disable= now accepts VM keyword to disable licenses on Virtual Machines.
Refresh buttons added to web interface.
options= license attribute.
Multiple instances of a single ISV-defined hostid type allowed.
License administration NOPROJECT keyword for EXCLUDE and EXCLUDEALL.

Table 6.30. v4.0 - December, 2007

Features Added:
Report log anonymizer (rlmanon).
RLM web interface allows editing option files.
RLM web interface displays debug log.
Report log detailed format adds seconds, tenths of seconds for denials.
Automatic report log rotation.
RLM options file controls access to administration functions.
RLM web interface displays recent debug log information.
RLM web interface allows editing server options file.
RLM_ROAM no longer needs to be set on the disconnected system.
-c overrides RLM_LICENSE for rlmutil.
Named User licensing.
disable=TerminalServer license attribute.
RLM activation supports issued=today's date.
RLM activation supports arbitrary license fields (via rlm_act_request() call).
RLM activation allows ISV-defined hostids as a legal hostid type.
RLM on Solaris supports Solaris containers.
rlm_all_hostids(), rlm_all_hostids_free() API calls.
rlm_get_attr_lfpath() API call.
rlm_set_active() API call.
rlm_license_exp_days() API call.
rlm_license_max_share() API call.
rlm_license_named_user_count() API call.
rlm_license_named_user_min_hours() API call.
rlm_license_roaming() API call.

Multiple ethernet device support on Linux and Mac.
Ethernet address is default hostid on Linux and Mac.
Windows volume serial number hostid added.
Windows volume serial number is default hostid.
Platforms Added:
PPC Linux platform.

Table 6.31. v3.0 - June, 2007

Features Added:
Internet Activation.
rlm -dat command-line option.
rlmtests performance tests.
RLM servers ignore hostnames in license file.
The RLM web interface now reports the Process ID (PID) of licenses in use.
RLM logs status requests in the debug log.
Client node can access license server by any name.
ISV server pathname optional on ISV line.
rlm_skip_isv_down() API call.
rlm_forget_isv_down() API call.
rlm_hostid() API call.
rlm_license_goodonce() API call.
rlm_license_server() API call.
rlm_product_exp() API call.
rlm_product_max_share() API call.
rlm_sign_license() API call.
x64_l1 compiled with -fPIC.
RLM_CONNECT_TIMEOUT environment variable.
RLM_EXTENDED_ERROR_MESSAGES environment variable.
Maximum license share count.
_line_item license keyword.
License in a string.
Improved error messages in web interface and rlmsign.
PID of process using license is displayed in web interface.
Wildcards allowed in IP addresses used as a hostid.
Platforms Added:
Java Linux
Java Windows

Table 6.32. v2.0 - Dec, 2006

Features Added:
Failover License Servers.
Token-based licensing.
user/host based licenses.
Nodelocked, single-use licenses (no server).
Options to disable rlmdown and rlmremove.
RLM_PATH_RANDOMIZE environment variable.
ISV servers notify of licenses expiring within 14 days.
RLM binds all TCP/IP ports in all license files.
rlm -c license_file command-line option.

RLM runs as a service on Windows.
rlmstat -avail reports on license availability.
Transient attribute on ISV-defined hostids.
System Info in RLM web interface.
min_remove license keyword.
rlm_products() API call.
rlm_log(), rlm_dlog() API calls.
PRIORITY license administration option.
TIMEZONE license administration option.
MAX accepts '*' for all users.
License administration license management by PROJECT.
MINREMOVE license administration option.
Platforms Added:
HP-UX PA-Risc 32-bit
HP-UX PA-Risc 64-bit
IBM AIX RS/6000 32-bit
IBM AIX RS/6000 64-bit

Table 6.33. v1.1 - July, 2006

Features Added:
Held licenses.
Shared licenses.
License Replacement.
License timeout.
Roaming licenses.
Intelligent license queuing.
rlm_set_environment() call.
rlm_license_XXX() calls.
rlm_auto_hb() call for automatic heartbeats.
ISV-defined hostids.
Clock rollback detection.
contract= license attribute.
customer= license attribute.
issued= license attribute.
issuer= license attribute.
platforms= license attribute.
soft_limit= license attribute.
start_date= license attribute.
timezone= license attribute.
type= license attribute.
Platforms Added:
MAC intel 32-bit
MAC PPC 32-bit
Linux x64 64-bit
Solaris x64 64-bit
Windows x64 64-bit

Table 6.34. v1.0 - May, 2006

Features:
Node-locked licenses.
Floating licenses.
Expiration dates.
Transparent multiple server connections.
Public-Key authentication.
Platforms Added:
Linux x86
Sun Solaris 32-bit
Sun Solaris 64-bit
Windows 32-bit

6.7 Appendix H - RLM Temporary Files

RLM stores roam files, server lockfiles, 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) (server lockfiles still in /var/tmp)
- ProgramDataReprise (Windows)
- Docs and SettingsAll UsersApplication DataReprise (Windows pre-vista).

Files stored here include:

- server lock files
- server roam files
- server transfer definitions
- server meter files
- client-side detached demo files
- client-side roam files
- client-side cache files
- client-side "single" lockfiles
- client-side rehostable hostid data
- client-side temporary licenses from RLM Cloud
- ASH contact data

The server's lock file name is rlmlock<*isv*> on Windows, and .rlmlock<*isv*> on Linux/Mac, where <*isv*> is your ISV name.

If you have lockfile problems on Linux which can't be attributed to another copy of the server running, then check the protection on /var/tmp to make sure it allows world rwx. On Windows, what sometimes happens when the first version of the RLM server to run on a system is pre-v9.4, is that the lock file is created with restrictive permissions that don't allow a different user to lock it. V9.4 and later create the Reprise folder with wide-open access that is inherited by files created in it, so this problem doesn't occur.

Deleting the lock file is OK to solve the immediate problem.

The server's roam file name is:

isvname&product&version&share&max_share&hostid

(on Unix systems, this name is preceded with a single ".")

The rehostable hostids are directories inside <rlm directory>/isvname

The ASH contact data is contained in <rlm directory>isvname.last on Windows and <rlm directory>/.isvname.last on non-Windows.

6.7.1 Notes on the v14.1 filename transition on Mac

In the transition in v14.1, RLM will continue to look in the old location (/var/tmp) for server lock files and client-side "single" lockfiles. RLM will look in the new location for all other files listed above. meaning that roaming/metering/detached demo/transfer 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. Finally, client-side temporary licenses and ASH contact data, which are both new in v14.1, will be in the new directory.

6.7.2 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^{tm} files
- Client-side roam files
- Client-side cache files
- Client-side "single" lockfiles
- Client-side rehostable hostid data
- Client-side temporary licenses from RLM Cloud (new in 14.1)
- ASH contact data (new in 14.1)

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.