

## **RLM Heartbeat Tutorial**

### **March 2009**

#### **Introduction**

Heartbeats are messages sent from a licensed application to the license server while the application has one or more licenses checked out from the server. The license server acknowledges the receipt of each heartbeat by sending a message back to the application. In this way, each side can know that the other side is up, running, and healthy, or take appropriate action if the other side is not healthy.

Not all licensed applications use heartbeats, and by default they don't. The application developer must make specific RLM function calls to make use of heartbeats. Typically short duty cycle applications – those that run only for a short period of time, like compilers – don't use heartbeats. It would be pointless to use heartbeats in an application whose duty cycle is on the order of a couple of minutes or less.

Note that if a licensed application (client) crashes or exits without checking in its licenses, the license server detects this and returns any licenses used by the crashed or exited application to the pool of available licenses. However, if the client operating system crashes or the network between them is segmented, the server is unaware of this and the client's licenses remain checked out until the server times the licenses out or human intervention is taken.

#### **ISV server TIMEOUT**

The ISV server keeps track of how long it has been since it has received a heartbeat from each client. The end user can set up a TIMEOUT (per product) or TIMEOUTALL (all products) in the ISV options file. For each client with a license checked out, if the server hasn't received a heartbeat within the specified interval, it decides that the client is no longer active, and returns its license(s) to the available pool. By default the ISV server doesn't have any timeout. The ISV may specify on a per-license basis the minimum timeout that may be specified by the end user. This is done with the "min\_timeout=n" optional license attribute.

If a TIMEOUT interval is set too low and an active client's license is returned to the available pool, an appropriately-coded client will attempt to reacquire the license without interruption of service. See the discussion of reconnection in the automatic and manual heartbeat sections below.

#### **Using Heartbeats in the Client**

The licensed application developer has the choice between using automatic heartbeats, manual heartbeats, or no heartbeats at all (the default). With automatic heartbeats, RLM creates a thread to perform the sending of heartbeats and tracking of responses. With manual heartbeats, the client calls an RLM function periodically which checks for the acknowledgement of the previous heartbeat and sends another one. Automatic heartbeats are a good choice where the application can afford for RLM to create a thread. Where

the application needs all available threads for itself, manual heartbeats are the logical choice.

### **Automatic Heartbeats**

To set up automatic heartbeats, the client calls `rlm_auto_hb` any time after `rlm_init` is called successfully.

```
rlm_auto_hb(RLM_HANDLE handle, int period, int auto_reconn, void
(*handler)())
```

See the RLM Integrators guide for full details on return values and arguments.

The `period` argument specifies how often to send a heartbeat. A good nominal value is 60 seconds.

The `auto_reconn` argument specifies whether or not RLM should attempt to reconnect to the license server and re-checkout licenses if the connection to the server is lost. Most applications will specify that reconnection should be tried.

The `handler` argument is a callback function supplied by the developer, which is called by RLM after a reconnect attempt is made. The purpose of this callback is to inform the application that reconnect attempts are being made (which implies loss of communications with the server), so that the application can make decisions about whether to continue or not.

When the client's reconnect handler is called, the application should first determine if the reconnect attempt was successful, then decide if some further action is necessary. The application will have some policy built-in about how long a period it is willing to tolerate being disconnected from the license server before it takes some action. The action taken in that case is also up to the application. It might save any work in progress and exit, or it might enter a reduced functionality mode until its license is restored.

The application determines if the reconnect was successful by calling `rlm_license_stat()`. This function **may not** be called from the handler though, so the handler should set a flag indicating that another thread should do the check.

### **Manual Heartbeats**

If use of automatic heartbeats is not practical in a given application, manual heartbeats may be used instead. The function `rlm_get_attr_health()` deals with manual heartbeats. Note that the sending and receiving of heartbeats is asynchronous – the client does **not** send a heartbeat and wait for a response. Instead, `rlm_checkout()` sends the first heartbeat to the server. Then each call to `rlm_get_attr_health()` first checks to see if a response to the previous heartbeat has arrived, then sends a heartbeat. `rlm_get_attr_health` is used as follows:

```
int status = rlm_get_attr_health(RLM_LICENSE license)
```

The status returned indicates the health of the connection to the license server. 0 means OK, RLM\_EL\_NO\_HEARTBEAT means that the last heartbeat wasn't acknowledged by the server, and RLM\_EL\_SERVER\_DOWN means the server is down. See license.h for full details on status values.

The application should call `rlm_get_attr_health()` every minute or two. RLM keeps track internally of the time of last call to `rlm_get_attr_health()`, and blocks the sending of heartbeats more frequently than once every 30 seconds so as not to use network resources irresponsibly. The application developer decides where in the code to call `rlm_get_attr_health()`. For an interactive application, a good choice is the main loop that accepts and processes user input.

Just as is the case with applications using automatic heartbeats, the application using manual heartbeats will have some policy as to how long it will tolerate unacknowledged heartbeats, and what action to take when that threshold is exceeded.

Unlike automatic heartbeats, there is no automatic reconnection with manual heartbeats. An application which receives an error from `rlm_get_attr_health()` should call `rlm_checkin()` and retry `rlm_checkout()`.

### **Heartbeats and Uncounted or Single Licenses**

It is tempting to think that an application that uses only uncounted licenses needn't be concerned with heartbeats, but this is not the case. There are 2 reasons for this:

1. If an application is enabled with a license locked to a removable host ID (RLMID1 or RLMID2), the application needs to know if and when that removable host is no longer present. It does this via the heartbeat mechanism. `rlm_license_stat()` and `rlm_get_attr_health()` return `RLM_EL_PORTABLE_REMOVED` in this case.
2. Very often, applications are developed with one kind of license in mind, eg, nodelocked uncounted. But after deployment an unanticipated situation often arises that dictates another kind of license should be used. If the ability to handle heartbeats is coded into the application, then the style of license available at runtime is transparent to the application - no changes to the application are necessary when a new license type is deployed.