Titanium Server

Heartbeat Service

Host-Guest Message-Based API
&
Guest Reference Implementation

Release: 15.12

01/14/2016

WIND RIVER

[Specification of the Host-Guest Heartbeat Service APIs – 15.12.]

Copyright Notice

Table of Contents

Introduction

Titanium Server implements a Host-to-Guest Heartbeat Service Messaging API to monitor the health of guest application(s) within a hosted VM. Loss of heartbeat will result in a corrective action being taken against the VM. The heartbeat interval and corrective action is specified by the VM.

Titanium Server's Heartbeat Service Messaging API also provides guest application(s) within a hosted VM, the ability to receive notification of and vote to accept or reject actions about to be performed against the VM. On notifications, the guest application within the VM can take this opportunity to cleanly shut down or transfer its service to a peer VM.

This document contains the specification for this messaging-based API.

Also included in this document is an overview of the Titanium Server Guest Heartbeat Service SDK Module which provides a Linux-based reference implementation of the Guest-side software for implementing this Messaging-based API in the Guest. The SDK Module provides source code and make/build instructions which can be used strictly as reference or built and included 'as is' in your Guest image. Full build, install and usage instructions can be found in the README files included in the SDK Module. This document simply provides an overview of the reference implementation.

Host – Guest Heartbeat Service Messaging API

Titanium Server implements a Host-to-Guest Heartbeat Service Messaging API to monitor the health of guest application(s) within a hosted VM, and to provide guest application(s) within a hosted VM, the ability to receive notification of and vote to accept or reject actions about to be performed against the VM.

The Host-to-Guest Heartbeat Service Messaging API is a message-based API using a JSON-formatted application messaging layer on top of a 'virtio serial device' between QEMU on the host and the Guest VM. JSON formatting provides a simple, humanly readable messaging format which can be easily parsed and formatted using any high level programming language being used in the Guest VM (e.g. C, Python, Java, etc.). Use of the 'virtio serial device' provides a simple, direct communication channel between host and guest which is independent of the Guest's L2/L3 networking.

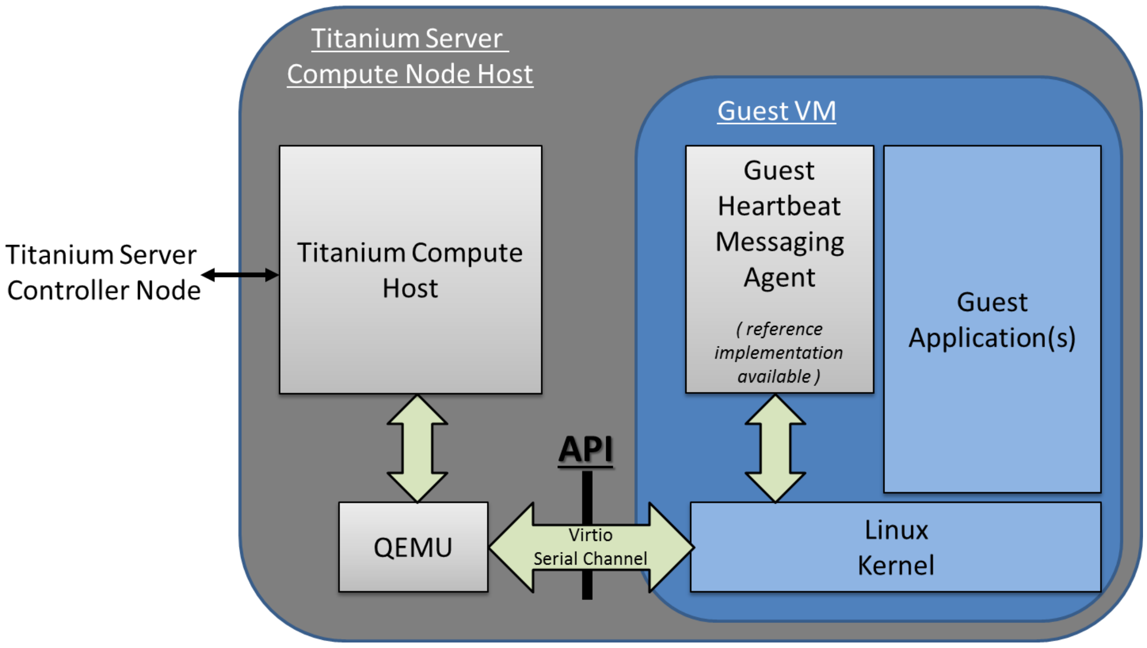
Titanium Server Compute Node Host
Guest VM
Titanium Server Controller Node
Titanium Compute Host
Guest Heartbeat Messaging Agent
(reference implementation available)
Guest Application(s)
API
QEMU
Virtio Serial Channel
Linux Kernel

Figure 1 – Titanium Server Host-Guest Heartbeat Service Messaging API

Message Types and Semantics

For the Heartbeat Messaging API, there are four message types; Init and Exit Connection Messages, Challenge / Response Messages, Action Notify / Response Messages and a Nack Message:

- Init (Guest → Host)
  - The Init message establishes the Guest – Host Heartbeat Messaging connection.
  - It is the Guest's responsibility to initialize the connection by sending the Init message to the Host,
  - A number of configurable parameters must be specified by the Guest when sending the Init message; e.g. the interval between heartbeat challenges from Host, the corrective action to take when heartbeat failure is detected, the timeout for accept/reject response, etc. ,
- Init Ack (Host → Guest)
  - This is the Host's successful acknowledgement of the Guest's Init request,
  - After receiving the Init Ack, the Guest must be prepared to respond to Challenge or Action Notify Requests,
- Exit (Guest → Host) → (Host → Guest)
  - The Exit message gracefully terminates the Guest – Host Heartbeat Messaging connection,
  - Either the Host or the Guest can send the Exit message,

- Challenge Request (Host → Guest)
    - The Challenge Request message is the periodic heartbeating of the Guest by the Host,
    - The Guest must respond within heartbeat interval specified by the Guest in the Init message, otherwise the Guest Heartbeat will be declared failed and the corrective action (reboot, stop, log) specified by the Guest in the Init message will be executed,

- Challenge Response (Guest → Host)
    - The Challenge Response is the Guest's response to the Host's heartbeat challenge,
    - The Guest must return
        - the sequence number and the heartbeat challenge (random) integer from the Challenge Request, and
        - the Guest health boolean (healthy, unhealthy), the desired corrective action if unhealthy, and a log message,

- Action Notify Request (Host → Guest)

    - The Action Notify Request message notifies the Guest of events being taken by the Host that will impact this Guest VM,
    - Actions include: stop, reboot, pause, unpause, suspend, resume, resize begin/end, live-migrate begin/end, cold-migrate begin/end,
    - There are two types of notification types
        - irrevocable (notification)
            - A notification that the specified 'action' will occur,
            - The Guest can execute any actions in order to gracefully prepare for the forthcoming action (e.g. sync state with peer, terminate protocol connections, flush and close files, etc.) …
              and respond with the Action Notify Response when complete,
            - If the Guest does not respond in the 'timeout' specified in the request, the action will proceed,
        - revocable (request)
            - Affectively a request as to whether the Guest would like to accept or reject an action,
            - E.g. the Guest may not be in a 'safe' state for the action to occur;
                - e.g. the Guest may be the Active-Side of a 1:1 Application which is not in-sync with its Standby-Side, therefore should reject a 'stop' action,
            - Guest responds with 'accept' or 'reject' in the Action Notify Response Message,
            - If the Guest does not respond in the 'timeout' specified in the request, 'accept' will be assumed,
            - If a revocable notification type is accepted,
              an irrevocable notification type will be sent for the Guest to gracefully prepare for the action.

- Action Notify Response (Guest → Host)

    - The Action Notify Response is the Guest's response to the Host's Action Notify Request,
    - The Guest must return
        - the sequence number, invocation ID, event type and notification type from the request, and
        - for the 'revocable' request case,
            - the vote result (accept or reject) and a log message.

- Nack (Host → Guest)
    - This is a message sent from the Host to the Guest when the Host receives a message with incorrect syntax,
    - It contains the message type of the original (incorrect) message and a log_msg describing the error,
    - This allows the Guest Application developer to debug issues when developing the Guest-side API code.

Virtio Serial Device

The transport layer of the Host-Guest Heartbeat Service Messaging API is a ‘virtio serial device’ (also known as a ‘vmchannel’) between QEMU (on the host) and the Guest VM. Device emulation in QEMU presents a virtio-pci device to the Guest, and a Guest Driver presents a char device interface to Guest userspace applications. This provides a simple transport mechanism for communication between the host userspace and the guest userspace. I.e. it is completely independent of the networking stack of the Guest, and is available very early in the boot sequence of the Guest.

This is a standard Linux QEMU/KVM feature. The Guest API for interfacing with the ‘virtio serial device’ can be found at http://www.linux-kvm.org/page/Virtio-serial_API . Examples of Guest code for opening, reading, writing, etc. from/to a ‘virtio serial device’ can also be found in the source code of the Titanium Server ‘Heartbeat Service Messaging SDK Module’. This SDK Module provides a Linux-based reference implementation of the Guest-side software for implementing the Guest Heartbeat Service Messaging API. Generally communicating with a ‘virtio serial device’ is very similar to communicating via a pipe, or a SOCK_STREAM socket.

There are however a few additional considerations to be aware of when using ‘virtio serial devices’:

- only one process at a time can open the device in the Guest,
- read() returns 0, if the Host is not connected to the device,
- write() blocks or returns -1 with error set to EAGAIN, if the Host is not connected,
- poll() will always set POLLHUP in revents when the Host connection is down.
  - This means that the only way to get event-driven notification of connection is to register for SIGIO. However, then a SIGIO event will occur every time the device becomes readable. The work-around is to selectively block SIGIO as long as the link is up is thought to be up, then unblock it on connection loss so a notification occurs when the link comes back.
- If the Host disconnects the Guest should still process any buffered messages from the device,
- Message boundaries are **not preserved**, the Guest needs to handle message fragment reassembly. Multiple messages can be returned in one read() call, as well as buffers beginning and ending with partial messages. This is hard to get perfect; one can study the `host_guest_msg.c` code in the Titanium Server Heartbeat Service Messaging SDK Module for ideas on how this can be handled.

The QEMU/KVM created by Titanium Server in order to host a Guest VM is created with a ‘virtio serial device’ named:

```
      /dev/virtio-port/cgcs.heartbeat
```

for Titanium Server Host – to – Guest VM Heartbeat Service Messaging.

JSON Message Syntax

The upper layer messaging format being used is ‘Line Delimited JSON Format’. I.e. a ‘\n’ character is used to identify message boundaries in the stream of data to/from the virtio serial device; specifically a ‘\n’ character is inserted at the start and end of the JSON Object representing a Message.

```
\n{key:value,key:value,…}\n
```

Note that key and values must NOT contain ‘\n’ characters.

Host – to – Guest Messages

Init Ack

Key	**Value**	**Optionality***	**Example value**	**Description**
“version”	integer	M	2	Version of the interface
“revision”	integer	M	1	Sub-version
“msg_type”	“init_ack”	M	“init_ack”	The type of message.
“sequence”	integer	M	9	Sequence Number; incremented by Host for each message sent.
“invocation_id”	integer	M	1714636915	Invocation ID from the received Init message.

- M: Mandatory; O: Optional; (Condition)

Exit

Key	**Value**	**Optionality***	**Example value**	**Description**
“version”	integer	M	1	Version of the interface
“revision”	integer	M	1	Sub-version
“msg_type”	“exit”	M	“exit”	The type of message.
“sequence”	integer	M	1	Sequence Number; incremented by Host for each message sent.
“log_msg”	string	M		Error message. Should not contain newline ‘\n’ character.

- M: Mandatory; O: Optional; (Condition)

WIND™

Challenge Request

Key	Value	Optionality*	Example value	Description
“version”	integer	M	2	Version of the interface
“revision”	integer	M	1	Sub-version
“msg_type”	“challenge”	M	“challenge”	The type of message.
“sequence”	integer	M	10	Sequence Number; incremented by Host for each message sent.
“heartbeat_challenge”	integer	M	1957747793	A random generated number

- M: Mandatory; O: Optional; (Condition)

Action Notify Request

Key	Value	Optionality*	Example value	Description
“version”	integer	M	2	Version of the interface
“revision”	integer	M	1	Sub-version
“msg_type”	“action_notify”	M	“action_notify”	The type of message.
“sequence”	integer	M	3	Sequence Number; incremented by Host for each message sent.
“invocation_id”	integer	M	1804289383	Invocation ID; unique transaction number for each request sent.
“event_type”	string “stop” “reboot” “suspend” “pause” “unpause” “resume” “resize_begin” “resize_end” “live_migration_begin” “live_migration_end” “cold_migration_begin” “cold_migration_end”	M	“pause”	Type of events
“notification_type”	string “revocable” “irrevocable”	M	“revocable”	Type of notification
“timeout_ms”	integer	M	9000	Timeout for notification in milli-seconds

- M: Mandatory; O: Optional; (Condition)

Nack

Key	Value	Optionality*	Example value	Description
“version”	integer	M	2	Version of the interface
“revision”	integer	M	1	Sub-version
“msg_type”	“nack”	M	“nack”	The type of message.
“sequence”	integer	M	2	Sequence of the message
“invocation_id”	integer	M	846930886	Invocation ID, equal to the one in previous received message from guest.
“orig_msg_type”	string	M	“init”	The type of message that host previous received from guest.
“log_msg”	string	M	“failed to parse version”	Error message

- M: Mandatory; O: Optional; (Condition)

Guest – to – Host Messages

Init

Key	Value	Optionality*	Example value	Description
“version”	integer	M	2	Version of the interface
“revision”	integer	M	1	Sub-version
“msg_type”	“init”	M	“init”	The type of message.
“sequence”	integer	M	2	Sequence Number; incremented by Guest for each message sent.
“invocation_id”	integer	M	846930886	Invocation ID; unique transaction number for each Init sent.
“name”	string	M	“vm1”	Name of the VM
“heartbeat_interval_ms”	integer	M	1000	Interval between heartbeat challenges in milli-seconds
“vote_secs”	integer	M	9	Timeout for vote in seconds
“shutdown_notice_secs”	integer	M	9	Timeout for shutdown notice in seconds
“suspend_notice_secs”	integer	M	9	Timeout for suspend notice in seconds
“resume_notice_secs”	integer	M	14	Timeout for resume notice in seconds
“restart_secs”	integer	M	601	Time for restart in seconds
“corrective_action”	String “reboot” “stop” “log”	M	“reboot”	Corrective action to take when heartbeat failure detected

- M: Mandatory; O: Optional; (Condition)

Exit

Key	Value	Optionality*	Example value	Description
“version”	integer	M	1	Version of the interface
“revision”	integer	M	1	Sub-version
“msg_type”	“exit”	M	“exit”	The type of message.
“sequence”	integer	M	1	Sequence Number; incremented by Guest for each message sent.
“log_msg”	string	M		Error message. Should not contain newline ‘\n’ character.

- M: Mandatory; O: Optional; (Condition)

Challenge Response

Key	Value	Optionality*	Example value	Description
"version"	integer	M	1	Version of the interface
"revision"	integer	M	1	Sub-version
"msg_type"	"challenge_response"	M		The type of message.
"sequence"	integer	M	1	Sequence Number; incremented by Guest for each message sent.
"heartbeat_response"	integer	M	278722862	Value is the same as the value of "heartbeat_challenge" from the Challenge Request.
"heartbeat_health"	string "healthy" "unhealthy"	M	"healthy"	Indicate whether or not guest is healthy
"corrective_action"	string "reboot" "stop" "log"	O	"reboot"	Corrective action to take when heartbeat failure detected; if not specified the corrective_action from the Init message is used.
"log_msg"	string	M ("unhealthy")	"File /tmp/unhealthy does not exist."	Error message. Should not contain newline '\n' character.

- M: Mandatory; O: Optional; (Condition)

Action Response

Key	Value	Optionality*	Example value	Description
“version”	integer	M	2	Version of the interface
“revision”	integer	M	1	Sub-version
“msg_type”	“action_response”	M	“action_response”	The type of message.
“sequence”	integer	M	3	Sequence Number; incremented by Guest for each message sent.
“invocation_id”	integer	M	1804289383	Invocation ID from the received Action Request message.
“event_type”	string “stop” “reboot” “suspend” “pause” “unpause” “resume” “resize_begin” “resize_end” “live_migration_begin” “live_migration_end” “cold_migration_begin” “cold_migration_end”	M	“reboot”	Type of event
“notification_type”	string “revocable” “irrevocable”	M	“revocable”	Type of notification
“vote_result”	string “accept” “reject” “complete” “timeout” “error”	M	“accept”	Result of the vote
“log_msg”	string	M (“reject”, “timeout”, “error”)	“”	Reject reason if vote is rejected; Error message if vote_result is “error” or “timeout”.

- M: Mandatory; O: Optional; (Condition)

Examples

Examples of Heartbeat Service Messaging JSON messages.

Init, Init Ack, and Exit:

Guest sends an Init to TiS to establish the Heartbeat Service Messaging Connection:

```
\n{"version":2,"revision":1,"msg_type":"init","sequence":10,"invocation_id":8
46930886,"name":"vm1","heartbeat_interval_ms":1000,"vote_secs":9,"shutdown_no
tice_secs":9,"suspend_notice_secs":9,"resume_notice_secs":14,"restart_secs":6
01,"corrective_action":"reboot"}\n
```

TiS responds with an Init Ack:

```
\n{"version":2,"revision":1,"msg_type":"init_ack","sequence":29,"invoca
tion_id":846930886}\n
```

…

Either Guest or TiS sends an Exit to close the Heartbeat Service Messaging Connection:

```
\n{"version":2,"revision":1,"msg_type":"exit","sequence":11,"log_msg":"Guest
Shutting Down."}\n
```

Challenge Request and Response:

A Challenge Request sent by TiS to the Guest:

```
\n{"version":2,"revision":1,"msg_type":"challenge","sequence":12,"heartbeat_c
hallenge":278722862}\n
```

The corresponding response from the Guest:

```
\n{"version":2,"revision":1,"msg_type":"challenge_response","sequence":
30,"heartbeat_response":278722862,"heartbeat_health":"healthy","correct
ive_action":"reboot","log_msg":"File /tmp/unhealthy does not exist."}\n
```

Action Notify Request and Response:

An Action Notify Request sent by TiS to the Guest:

```
\n{"version":2,"revision":1,"msg_type":"action_notify","sequence":13,"invocat
ion_id":1804289383,"event_type":"pause","notification_type":"revocable","time
out_ms":9000}\n
```

The corresponding response from the Guest:

```
\n{"version":2,"revision":1,"msg_type":"action_response","sequence":31,
"invocation_id":1804289383,"event_type":"pause","notification_type":"ir
revocable","vote_result":"accept","log_msg":""}\n
```

Nack:

Guest sends an Init (with an invalid version) to TiS to establish the Heartbeat Service Messaging Connection:

```
\n{"version":"V2.0","revision":1,"msg_type":"init","sequence":10,"invocation_
id":846930886,"name":"vm1","heartbeat_interval_ms":1000,"vote_secs":9,"shutdo
wn_notice_secs":9,"suspend_notice_secs":9,"resume_notice_secs":14,"restart_se
cs":601,"corrective_action":"reboot"}\n
```

TiS responds Nack:

```
\n{"version":2,"revision":1,"msg_type":"nack","sequence":30,"invocation
_id":846930886,"orig_msg_type":"init","log_msg":"failed to parse
version"}\n
```

Reference Implementation of Guest Heartbeat Service Messaging

This section provides an overview of the Linux-based reference implementation of the Guest-side software for implementing this Host-to-Guest Heartbeat Service Messaging API in the Guest.

This reference implementation can be found in the Titanium Server Guest Heartbeat Service Messaging SDK Module. This Module provides source code and make/build instructions which can be used strictly as reference or built and included 'as is' in your Guest image. Full build, install and usage instructions can be found in the README files included in the SDK Module. This section simply provides an overview of the reference implementation.

The diagram below provides the architecture diagram of the reference implementation:

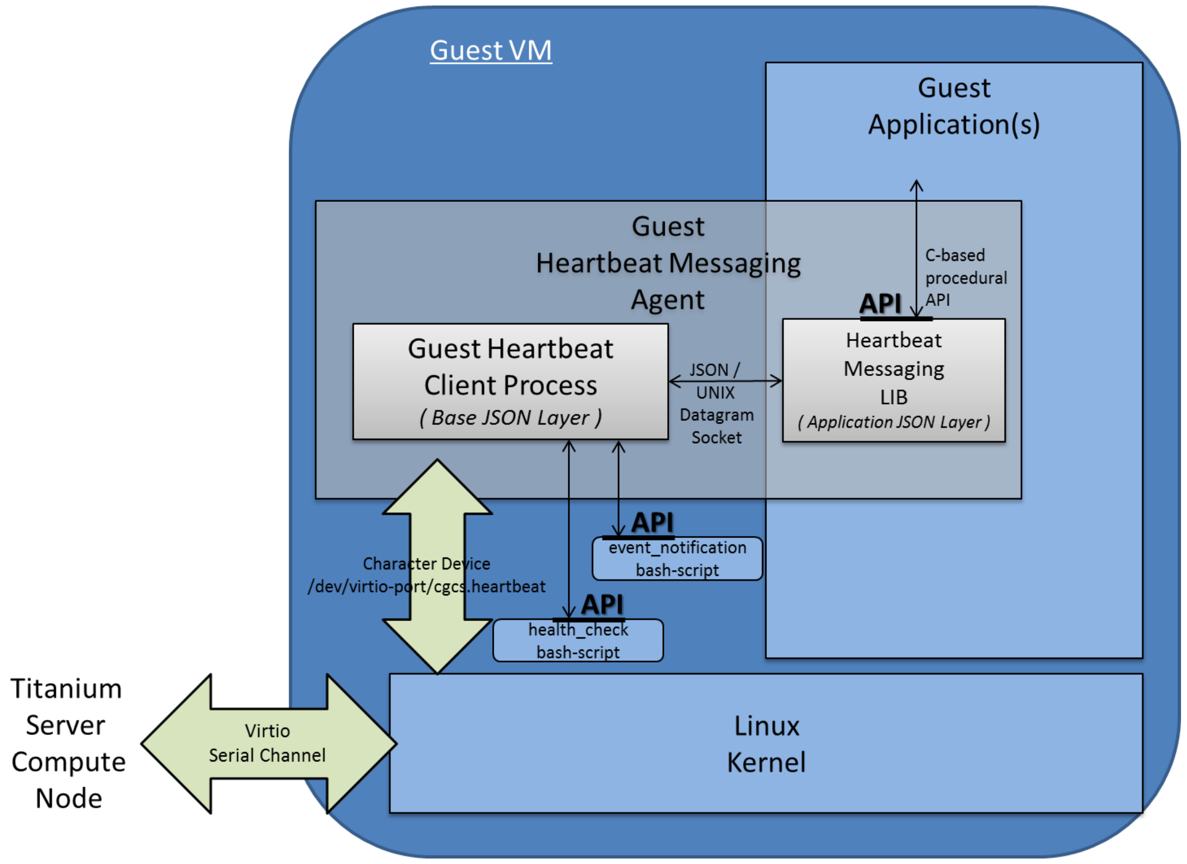
Guest VM
Guest Application(s)
Guest Heartbeat Messaging Agent
C-based procedural API
API
Guest Heartbeat Client Process
(Base JSON Layer)
JSON / UNIX Datagram Socket
Heartbeat Messaging LIB
(Application JSON Layer)
API
event_notification bash-script
Character Device /dev/virtio-port/cgcs.heartbeat
API
health_check bash-script
Titanium Server Compute Node
Virtio Serial Channel
Linux Kernel

Figure 2 – Reference Implementation Architecture for Guest Heartbeat Service Messaging

Where:

- A Guest Heartbeat Client Process provides a simple implementation of the Guest Side Heartbeat Service.
  This includes:
  - opening/reading,/writing and general management of the virtio serial device between the Guest and the Host,
  - initializing the Guest Heartbeat Service connection at init time by sending the Init message to the TiS Host
    - the values of the various configurable parameters in the Init message are taken from a /etc/guest-client/heartbeat /guest_heartbeat.conf configuration file,
    - see the Titanium Server Guest Heartbeat Service Messaging SDK Module for examples and full details on the format of the guest_heartbeat.conf file,
  - by default, the Guest Heartbeat Client Process will
    - respond as healthy to all TiS Host Challenge Requests,
    - respond with accept to all revocable TiS Host Action Notify Requests, and
    - respond with complete to all irrevocable TiS Host Action Notify Requests.
  - optionally, a health-check script and/or an event-notification script can be specified in the /etc/guest-client/heartbeat /guest_heartbeat.conf configuration file, in order to add Guest Application Specific behavior
    - health-check script
      - a script called with no arguments, and
      - exiting with 1 for unhealthy and 0 for healthy
    - event-notification script
      - a script called with two arguments
        - revocable or irrevocable notification type, and
        - the particular event type, e.g. stop, reboot, pause, etc.
      - in the revocable case,
        - exiting with 1 for reject and 0 for accept
      - in the irrevocable case,
        - doing the appropriate preparations for gracefully preparing for the action.

    Again, see the Titanium Server Guest Heartbeat Service Messaging SDK Module README and source files for examples and full details.

- The Guest Heartbeat Client Process also extends the service by allowing one or more VM resident application specific processes, to register for heartbeating and event notification.
  - Each process can specify its own heartbeat interval, and its own corrective action to be taken against the VM as a whole,

    - Each process can participate in the accept/reject voting for a an action notification (must be unanimous) and be notified for graceful preparations of an upcoming event
    - The interface between the Guest Heartbeat Client Process and the additional registered Applications:
        - is a message-based interface;
        - specifically a JSON Messaging Layer over a UNIX Datagram socket,

- A Heartbeat Messaging Lib provides a C-based procedural API for a registering Application Process to interface with the Guest Heartbeat Client Process.

  Specifically this library implements the interface described above; a JSON Messaging Layer over a UNIX Datagram socket, with a C-based procedural API.

  Again, see the Titanium Server Guest Heartbeat Service Messaging SDK Module README and source files for full details of this C-based procedural API and example usage.