Guest and HA Application Monitoring Developer's Guide

vSphere Guest SDK 9.0 and
vSphere HA Application Monitoring
for vSphere 5.5

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.

EN-001156-00

vmware®

You can find the most up-to-date technical documentation on the VMware Web site at:

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to:

docfeedback@vmware.com

Copyright © 2005–2013 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at http://www.vmware.com/go/patents.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About This Book

The *Guest and HA Application Monitoring Developer's Guide* provides information about developing applications using the VMware® Guest Application Programming Interface (API).

VMware provides several different software development kit (SDK) products, each of which targets different developer communities and platforms. This guide is intended for developers who want to retrieve information about the virtual machine and host hardware in which the application runs. The supported VMware platforms include ESX/ESXi 4.0, ESX/ESXi 4.1, ESXi 5.0, ESXi 5.1, and ESXi 5.5.

Revision History

This book is revised with each release of the product or when necessary. A revised version can contain minor or major changes. Table 1 summarizes the significant changes in each version of this book.

Table 1. Revision History

Revision Date	Description
19 Sept 2013	Update for ESXi 5.5, with new VMGuestAppMonitor_PostAppState function.
17 May 2012	Added vSphere HA Application Monitoring, changed version number for VMware Tools 9.0.
24 Aug 2011	Added information about compatibility with vSphere 5.0.
13 Jul 2010	No new information, but revised to note support for VMware ESX 4.1.
7 May 2009	Revised manual for VMware ESX version 4.0.
29 Nov 2007	No new information, but revised to note support for VMware ESX 3.5 and ESX 3i version 3.5.
18 Jul 2005	Initial release of the VMware Guest SDK providing support for VMware ESX 3.0.

Intended Audience

This book is intended for developers of software for vSphere high availability (HA) application monitoring, or for gathering statistics about guest operating systems.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation go to http://www.vmware.com/support/pubs.

Document Feedback

VMware welcomes your suggestions for improving our documentation. Send your feedback to docfeedback@vmware.com.

Installing the Development Kit 1

Welcome to the VMware Guest and High Availability (HA) Application Monitoring software development kit. This chapter covers the following topics:

About the SDK Contents

The Guest and HA Application Monitoring SDK is available as a tarball for Linux or a ZIP file for Windows. Both have a similar directory structure, shown in Table 1-1, with minor differences for compilation.

Table 1-1. Components of the SDK

Directory or Folder	Explanation of Contents
`bin/bin32` or `bin/win32` `bin/bin64` or `bin/win64`	Contains the `vmware-appmonitor` program, which controls the HA application monitoring heartbeat from the command line.
`docs/`	Contains the Guest SDK terms and conditions, and text of related Open Source licenses. Also contains sample code for HA application monitoring.

`docs/VMGuestAppMonitor/`
`samples/C` or
`samples/visualstudio`
`samples/java`
	The `samples/C` subdirectory (or `samples/visualstudio` subfolder) contains the `sample.c` (or `appmon.cpp`) program to demonstrate HA application monitoring API. Follow instructions in the README file to compile with `make` or with Visual Studio. The `samples/java` directory contains a Java native interface (JNI) implementation that builds on the C implementation. Again, see the README file.

`include/`	Header files for include, basic types, the GuestAppMonitor library, the Guest library, and session ID.
`include/vmGuestLibTest.c`	Sample C program to run all the Guest library functions and return statistics. On Linux, use `gcc` to compile this program, and run it on an ESXi hosted virtual machine.
`lib/lib32` or `lib/win32` `lib/lib64` or `lib/win64`	Shared objects or DLL files and libraries for the Guest library, the Guest library for Java, and the HA application monitoring library.
`vmGuestLibJava` `vmGuestLibJava/doc`	JAR file and standard Javadoc for a prepackaged Java implementation of the Guest API. For a list of methods, browse `index.html` and see the VMGuestLibInterface page.

Displaying vSphere Guest Library Statistics

On a Linux virtual machine hosted by ESX/ESXi 3.5 or later, go to the `include` directory and compile the `vmGuestLibTest.c` program. Then run the output program `vmguestlibtest`.

```
gcc -g -o vmguestlibtest -ldl vmGuestLibTest.c
./vmguestlibtest
```

Guest statistics appear repeatedly until you interrupt the program.

Using the HA Application Monitoring SDK

This section provides a short introduction to the HA Application Monitoring SDK. You need the information in Chapter 3, "vSphere HA Application Monitoring," on page 17 to proceed further.

SDK function definitions and simple documentation are in the `vmGuestAppMonitorLib.h` include file.

Controlling the Application Monitoring Heartbeat

To run HA application monitoring programs, the virtual machine's host must be running ESX/ESXi 4.1 or later, and application monitoring must have been enabled when configuring HA.

You can enable heartbeats with the precompiled `vmware-appmonitor` program. Usage is as follows:

```
vmware-appmonitor { enable | disable | markActive | isEnabled | getAppStatus }
```

- `enable` – Enable application heartbeat so vSphere HA starts listening and monitoring the heartbeat count from this guest virtual machine. The heartbeats should be sent at least once every 30 seconds.
- `disable` – Disable the application heartbeat so vSphere HA stos listening to heartbeats from this guest.
- `markActive` – This starts senting the actual heartbeat every 30 seconds or less.
- `isEnabled` – Indicates whether the heartbeating was enabled.
- `getAppStatus` – Gets the status of the application, either Green, Red, or Gray.

On Linux, set your `LD_LIBRARY_PATH` environment to the install location of `GuestSDK/lib/lib32` or `lib64`. On Windows, you can set your `PATH` environment, but it is probably easier to copy `vmware-appmonitor` to the same folder as the DLL files.

Compiling the Sample Program on Linux

You need a C compiler and the `make` program.

1. Go to the `docs/VMGuestAppMonitor/samples/C` directory.
2. Run the `make` command.

   On a 64-bit machine you might want to change `lib32` to `lib64` in the `makefile`.
3. Set `LD_LIBRARY_PATH` as described above.
4. Run the sample program. See below for program usage.

   ```
   ./sample
   ```

Compiling Sample Programs on Windows

You need Visual Studio 2008 or later.

1. Go to the `docs/VMGuestAppMonitor/samples/visualstudio` folder.
2. Open the `appmon.vcproj` file and build the solution.
3. Click **Debug > Start Debugging** to run `appmon.exe`. See below for program usage.

Demonstrating the HA Application Monitoring API

The sample program enables HA application monitoring and sends a heartbeat every 15 seconds. Once the program is running, typing Ctrl+C displays three choices:

- s – stop sending heartbeats and exit the program. This should cause a reset of the virtual machine.
- `d` – disable application monitoring and exit the program. This does not cause a reset.
- `c` – continue sending heartbeats.

The Guest Programming API 2

The VMware Guest API provides functions that you can use in a program that runs in the guest operating system environment on a VMware ESX/ESXi host. This guide includes the following topics:

Overview of the vSphere Guest API

The vSphere Guest API provides functions that management agents and other software can use to collect data about the state and performance of a VMware virtual machine. The Guest API provides fast access to resource management information, without the need for authentication.

The Guest API provides read-only access. You can read data using the API, but you cannot send control commands. To issue control commands, use the vSphere Web Services SDK. For more information, see the *VMware vSphere Web Services SDK Programming Guide* and the *VMware vSphere API Reference*, which are available on the VMware developer support Web site.

The version number of this Guest API release is 9.0 to match the version number of VMware Tools.

Supported Guest Operating Systems

The vSphere Guest API software runs on most Windows or Linux guest operating systems supported by the various versions of ESX and ESXi.

See the *VMware Compatibility Guide* for a list of supported guest operating system versions. This guide is now located at http://www.vmware.com/resources/compatibility in Web format.

The Guest API does not support the following guest operating system environments:

- Windows 95 and Windows 98.
- Windows NT 4.0. For Windows NT 4.0 you must use Guest SDK 3.5, which you can find by going to http://www.vmware.com/support/developer/guest-sdk and selecting an old release.

Virtual Machine Statistics

With the Guest API, you can monitor various statistics about the virtual machine. You can use this information to retrieve scheduling and resource usage information about the environment. With the help of these statistics, a virtual machine can immediately react to changes in its virtual environment at the application layer.

The following list shows some of the information that you can retrieve through the vSphere Guest API:

- Amount of memory reserved for the virtual machine.
- Amount of memory being used by the virtual machine.
- Upper limit of memory available to the virtual machine.

- Number of memory shares assigned to the virtual machine.
- Maximum speed to which the virtual machine's CPU is limited.
- Reserved rate at which the virtual machine is allowed to execute. An idling virtual machine might consume CPU cycles at a much lower rate.
- Number of CPU shares assigned to the virtual machine.
- Elapsed time since the virtual machine was last powered on or reset.
- CPU time consumed by a particular virtual machine. When combined with other measurements, you can estimate how fast the virtual machine's CPUs are running compared to the host CPUs.

IMPORTANT The API uses a handle that provides access to the statistics. The handle also is a mechanism to determine whether the API can provide accurate information. (Certain events, such a migrating a virtual machine with VMotion™, temporarily make it impossible to provide accurate information.)

How to Use the vSphere Guest API

The vSphere Guest API defines functions and data types that you use to extract virtual machine data. This section covers the following topics:

vSphere Guest API Runtime Components

To use the vSphere Guest API, the runtime components must be installed in the guest operating system. The runtime components are dynamically loaded binary modules for 32-bit and 64-bit guests. When you install VMware Tools, the vSphere Guest API runtime components are installed automatically. You can also download them from http://www.vmware.com/download/sdk/guest_sdk.html.

To make the vSphere Guest API functions available to your program, use your program's standard methods to load the library.

- In a Windows guest operating system, the library file is `vmGuestLib.dll`. The import library file is `vmGuestLib.lib`.
- In a Linux guest operating system, the library file is `libvmGuestLib.so`.

IMPORTANT If you are using a Security-Enhanced Linux (SELinux) guest operating system, the security policies might interfere with dynamic loading of `libvmGuestLib.so`. Refer to documentation about SELinux policy configuration.

The vSphere Guest SDK includes the test program `vmGuestlibTest.c`. If you are using a Windows environment, you must rebuild the test program. The `vmGuestLib.dll` library file is a non-Unicode DLL. In Microsoft Visual Studio, build the test program `vmGuestlibTest.c` as a non-Unicode executable so that the program can access the DLL at runtime.

Enabling and Disabling the Runtime Components

The vSphere Guest API runtime components are enabled by default (`disable = "FALSE"`). To disable the runtime components, use the configuration editor in the vSphere Client to edit the configuration file for the virtual machine. The virtual machine must be powered off before you can use the configuration editor.

1 In the vSphere Client window, right-click the virtual machine in the machine list.

2 In the drop-down menu, select **Edit Settings**.

3 In the Virtual Machine Properties window, click the **Options** tab.

4 In the list of "Advanced" settings, select **General**.

5 Click **Configuration Parameters**.

6 In the Configuration Parameters window, add the following line or, if the file already contains the disable configuration setting, set the value to **TRUE**:

```
isolation.tools.guestlibGetInfo.disable = "TRUE"
```

The default value for the disable setting is **FALSE**. The default setting enables the runtime components. Reinstalling VMware Tools does not affect the disable setting. If you disable the vSphere Guest API and then reinstall VMware Tools, the vSphere Guest API continues to be unavailable until you change the `guestLibGetInfo.disable` configuration setting to **FALSE**.

vSphere Guest API Data Types

The vSphere Guest API uses the data types listed in Table 1 to support access to virtual machine data.

Table 1. Data Types

Data Type	Description
`VMGuestLibHandle`	Reference to virtual machine data. `VMGuestLibHandle` is defined in `vmGuestLib.h`.
`VMSessionID`	Unique identifier for a session. The session ID changes after a virtual machine is migrated using VMotion, suspended and resumed, or reverted to a snapshot. Any of these events is likely to render any information retrieved with this API invalid. You can use the session ID to detect those events and react accordingly. For example, you can refresh and reset any state that relies on the validity of previously retrieved information. Use `VMGuestLib_GetSessionId` to obtain a valid session ID. A session ID is opaque. You cannot compare a virtual machine session ID with the session IDs from any other virtual machines.You must always call `VMGuestLib_GetSessionId` after calling `VMGuestLib_UpdateInfo`. `VMSessionID` is defined in `vmSessionId.h`.
`VMGuestLibError`	Status code that indicates success or failure. Each function returns a `VMGuestLibError` code. For information about specific error codes, see "vSphere Guest API Error Codes " on page 15. `VMGuestLibError` is an enumerated type defined in `vmGuestLib.h`.

vSphere Guest API Functions

The vSphere Guest SDK contains the header file `vmGuestLib.h`. This file declares the functions and data types that you use to call the vSphere Guest API. The following sections describe the vSphere Guest API functions:

- "Context Functions " on page 11
- "Accessor Functions (Virtual Machine)" on page 13

Context Functions

The vSphere Guest API provides a set of functions that initialize and manipulate the context in which the Guest API operates. Before your application can use the accessor functions to retrieve information about a virtual machine, use the following functions to initialize the vSphere Guest API environment.

1 Call the `VMGuestLib_OpenHandle` function to obtain a handle for accessing information about the virtual machine. The guest library handle is a parameter to every Guest API function.

2 Call the `VMGuestLib_UpdateInfo` function to update the information available through the handle.

3 Call the `VMGuestLib_GetSessionId` function to retrieve a session ID.

Example 1 shows a C code fragment that illustrates the function calls for initialization. (The code fragments in this section do not perform error handling. For information about error handling, see "vSphere Guest API Error Codes " on page 15.)

Example 1. Initializing the vSphere Guest API Environment

```
VMGuestLibHandle glHandle;
VMGuestLibError glError;
VMSessionId sid = 0;
glError = VMGuestLib_OpenHandle(&glHandle);
glError = VMGuestLib_UpdateInfo(glHandle);
glError = VMGuestLib_GetSessionId(glHandle, &sid);
```

You can use the session ID to detect changes that invalidate previously retrieved data. Example 2 shows a code fragment that illustrates how to use the session ID to detect stale data. (The `ResetStats` function in the following fragment represents application code to handle the session change.)

Example 2. Detecting Stale Data

```
VMGuestLibHandle glHandle;
VMGuestLibError glError;
VMSessionId oldSid;
VMSessionId newSid;

/* [...code here would access data based on an existing, valid session ID (oldSid)...] */

/* Update the library, get the session ID, and compare it to the previous session ID */
glError = VMGuestLib_UpdateInfo(glHandle);
glError = GuestLib_GetSessionId(glHandle, &newSid);
if (oldSid != newSid) {
    ResetStats();
    oldSid = newSid;
}
```

Table 2 lists the context functions for creating and releasing handles, updating information, and obtaining session IDs.

Table 2. Open, Close, and Update Functions

Function	Description
`VMGuestLib_OpenHandle`	Gets a handle for use with other vSphere Guest API functions. The guest library handle provides a context for accessing information about the virtual machine. Virtual machine statistics and state data are associated with a particular guest library handle, so using one handle does not affect the data associated with another handle.
`VMGuestLib_CloseHandle`	Releases a handle acquired with `VMGuestLib_OpenHandle`.
`VMGuestLib_UpdateInfo`	Updates information about the virtual machine. This information is associated with the `VMGuestLibHandle`. `VMGuestLib_UpdateInfo` requires similar CPU resources to a system call and therefore can affect performance. If you are concerned about performance, minimize the number of calls to `VMGuestLib_UpdateInfo`. If your program uses multiple threads, each thread must use a different handle. Otherwise, you must implement a locking scheme around update calls. The vSphere Guest API does not implement internal locking around access with a handle.
`VMGuestLib_GetSessionId`	Retrieves the `VMSessionID` for the current session. Call this function after calling `VMGuestLib_UpdateInfo`. If `VMGuestLib_UpdateInfo` has never been called, `VMGuestLib_GetSessionId` returns `VMGUESTLIB_ERROR_NO_INFO`.

Accessor Functions (Virtual Machine)

Accessor functions retrieve information about a virtual machine. When you call an accessor function, you pass a guest library handle (type `VMGuestLibHandle`) to the function. If the function is successful, it returns the requested data as an output parameter. The function return value is an error code (type `VMGuestLibError`) that indicates success or failure. Example 3 shows a C code fragment that illustrates an example of calling `VMGuestLib_GetCpuLimitMHz` to retrieve the processor limit available to the virtual machine.

Example 3. Using an Accessor Function

```
uint32 cpuLimitMHz = 0;
glError = VMGuestLib_GetCpuLimitMHz(glHandle, &cpuLimitMHz);
```

When a call completes successfully, the function returns the value `VMGUESTLIB_ERROR_SUCCESS`. Unsuccessful calls return error codes that contain an appropriate description as part of the error code name. For details, see "vSphere Guest API Error Codes " on page 15.

Call `VMGuestLib_UpdateInfo` once to refresh all statistics before calling an accessor function or a series of accessor functions.

Table 3. Accessor Functions for Virtual Machine Data

Function	Description
`VMGuestLib_GetCpuLimitMHz`	Retrieves the upper limit of processor use in MHz available to the virtual machine. For information about setting the CPU limit, see "Limits and Reservations" on page 14.
`VMGuestLib_GetCpuReservationMHz`	Retrieves the minimum processing power in MHz reserved for the virtual machine. For information about setting a CPU reservation, see "Limits and Reservations" on page 14.
`VMGuestLib_GetCpuShares`	Retrieves the number of CPU shares allocated to the virtual machine. For information about how an ESX server uses CPU shares to manage virtual machine priority, see the *vSphere Resource Management Guide*.
`VMGuestLib_GetCpuStolenMs`	Retrieves the number of milliseconds that the virtual machine was in a ready state (able to transition to a run state), but was not scheduled to run.
`VMGuestLib_GetCpuUsedMs`	Retrieves the number of milliseconds during which the virtual machine has used the CPU. This value includes the time used by the guest operating system and the time used by virtualization code for tasks for this virtual machine. You can combine this value with the elapsed time (`VMGuestLib_GetElapsedMs`) to estimate the effective virtual machine CPU speed. This value is a subset of elapsedMs.
`VMGuestLib_GetElapsedMs`	Retrieves the number of milliseconds that have passed in the virtual machine since it last started running on the server. The count of elapsed time restarts each time the virtual machine is powered on, resumed, or migrated using VMotion. This value counts milliseconds, regardless of whether the virtual machine is using processing power during that time. You can combine this value with the CPU time used by the virtual machine (`VMGuestLib_GetCpuUsedMs`) to estimate the effective virtual machine CPU speed. cpuUsedMS is a subset of this value.
`VMGuestLib_GetHostProcessorSpeed`	Retrieves the speed of the ESX system's physical CPU in MHz.
`VMGuestLib_GetMemActiveMB`	Retrieves the amount of memory the virtual machine is actively using—its estimated working set size.
`VMGuestLib_GetMemBalloonedMB`	Retrieves the amount of memory that has been reclaimed from this virtual machine by the vSphere memory balloon driver (also referred to as the "vmmemctl" driver).
`VMGuestLib_GetMemLimitMB`	Retrieves the upper limit of memory that is available to the virtual machine. For information about setting a memory limit, see "Limits and Reservations" on page 14.
`VMGuestLib_GetMemMappedMB`	Retrieves the amount of memory that is allocated to the virtual machine. Memory that is ballooned, swapped, or has never been accessed is excluded.

Table 3. Accessor Functions for Virtual Machine Data (Continued)

`VMGuestLib_GetMemOverheadMB`	Retrieves the amount of "overhead" memory associated with this virtual machine that is currently consumed on the host system. Overhead memory is additional memory that is reserved for data structures required by the virtualization layer.
`VMGuestLib_GetMemReservationMB`	Retrieves the minimum amount of memory that is reserved for the virtual machine. For information about setting a memory reservation, see "Limits and Reservations" on page 14.
`VMGuestLib_GetMemSharedMB`	Retrieves the amount of physical memory associated with this virtual machine that is copy-on-write (COW) shared on the host.
`VMGuestLib_GetMemSharedSavedMB`	Retrieves the estimated amount of physical memory on the host saved from copy-on-write (COW) shared guest physical memory.
`VMGuestLib_GetMemShares`	Retrieves the number of memory shares allocated to the virtual machine. For information about how an ESX server uses memory shares to manage virtual machine priority, see the *vSphere Resource Management Guide*.
`VMGuestLib_GetMemSwappedMB`	Retrieves the amount of memory that has been reclaimed from this virtual machine by transparently swapping guest memory to disk.
`VMGuestLib_GetMemTargetSizeMB`	Retrieves the size of the target memory allocation for this virtual machine.
`VMGuestLib_GetMemUsedMB`	Retrieves the estimated amount of physical host memory currently consumed for this virtual machine's physical memory.

`VMGuestLib_GetResourcePoolPath`	Retrieves the path name of the resource pool to which the virtual machine belongs on the ESX system where it is running.
	VMGuestLib_GetResourcePoolPath uses an additional parameter to determine the size of the path name output string buffer.
	
	`VMGuestLibError VmGuestLib_GetResourcePoolPath(`
	`VMGuestLibHandle handle,`
	`size_t *bufferSize,`
	`char *pathBuffer );`
	
	`handle` is an input parameter specifying the guest library handle.
	`bufferSize` is an input/output parameter. It is a pointer to the size of the pathBuffer in bytes. If bufferSize is not large enough to accomodate the path (including a NULL terminator), the function returns VMGUESTLIB_ERROR_BUFFER_TOO_SMALL. In this case, the function uses the bufferSize parameter to return the number of bytes required for the string.
	
	`pathBuffer` is an output parameter. It is a pointer to a buffer that receives the resource pool path string. The path string is NULL-terminated.
	For information about using resource pools, see the *vSphere Resource Management Guide*.

For more information about ESX resource management, see the *vSphere Resource Management Guide*, available on the VMware Web site.

Limits and Reservations

You use the Guest API to retrieve information about limits and reservations for CPU and memory. To set limits and reservations, you can use either the vSphere Client or the vSphere API. Setting limits and reservations on a virtual machine ensures that resources are available to that machine and to other virtual machines that draw resources from the same resource pool.

To use the vSphere Client to set limits or reservations:

1 In the vSphere Client window, click on the **Resource Allocation** tab.

2 In either the CPU or Memory section, click **Edit**.

3 In the Virtual Machine Properties Window, click on the **Resources** tab.

4 Select either the CPU or Memory setting.

5 Use the slider controls to set limits or reservations.

For more information, see the help for the vSphere Client.

To use the vSphere API to set limits or reservations, call the **ReconfigVM_Task** method. In the method call, use the **VirtualMachineConfigSpec** data object to set the **cpuAllocation** or **memoryAllocation** property. These properties are of type **ResourceAllocationInfo** type, which has **limit** and **reservation** properties. For more information, see the VMware vSphere API Reference Documentation.

vSphere Guest API Error Codes

Each vSphere Guest API function returns an error code. If successful, the returned error code is `VMGUESTLIB_ERROR_SUCCESS`. If the function is unable to complete its task, the error code provides information for diagnosing the problem.

Use the `VMGuestLib_GetErrorText` function to retrieve the error text associated with a particular error code. When you call the function, pass the error code to the function; `VMGuestLib_GetErrorText` returns a pointer to a string containing the error text.

Example 4 shows error handling. The C code fragment declares a guest library handle and calls the function `VMGuestLib_OpenHandle`. If the call is not successful, the code calls `VMGuestLib_GetErrorText` and displays the error text.

Example 4. Error Handling

```
VMGuestLibHandle glHandle;
glError = VMGuestLib_OpenHandle(&glHandle);
if (glError != VMGUESTLIB_ERROR_SUCCESS) {
   printf("OpenHandle failed: %s\n", VMGuestLib_GetErrorText(glError));
}
```

Table 4 lists all error codes defined for the vSphere Guest API.

Table 4. Error Codes

Error Code	Description
`VMGUESTLIB_ERROR_SUCCESS`	The function has completed successfully.
`VMGUESTLIB_ERROR_OTHER`	An error has occurred. No additional information about the type of error is available.
`VMGUESTLIB_ERROR_NOT_RUNNING_IN_VM`	The program making this call is not running on a VMware virtual machine.
`VMGUESTLIB_ERROR_NOT_ENABLED`	The vSphere Guest API is not enabled on this host, so these functions cannot be used. For information about how to enable the library, see "Context Functions " on page 11.
`VMGUESTLIB_ERROR_NOT_AVAILABLE`	The information requested is not available on this host.
`VMGUESTLIB_ERROR_NO_INFO`	The handle data structure does not contain any information. You must call `VMGuestLib_UpdateInfo` to update the data structure.
`VMGUESTLIB_ERROR_MEMORY`	There is not enough memory available to complete the call.
`VMGUESTLIB_ERROR_BUFFER_TOO_SMALL`	The buffer is too small to accommodate the function call. For example, when you call `VMGuestLib_GetResourcePoolPath`, if the path buffer is too small for the resulting resource pool path, the function returns this error. To resolve this error, allocate a larger buffer.
`VMGUESTLIB_ERROR_INVALID_HANDLE`	The handle that you used is invalid. Make sure that you have the correct handle and that it is open. It might be necessary to create a new handle using `VMGuestLib_OpenHandle`.
`VMGUESTLIB_ERROR_INVALID_ARG`	One or more of the arguments passed to the function were invalid.
`VMGUESTLIB_ERROR_UNSUPPORTED_VERSION`	The host does not support the requested statistic.

vSphere HA Application Monitoring 3

This chapter discusses the vSphere High Availability (HA) Application Monitoring and the following topics:

About vSphere HA

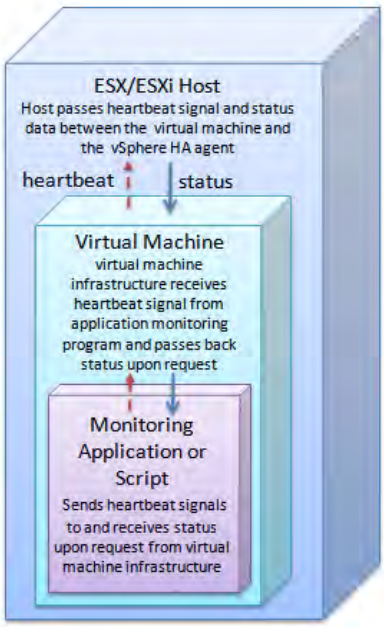
The vSphere High Availability (HA) feature for ESXi hosts in a cluster provides protection for a guest OS and its applications, by restarting the virtual machine if a guest OS or application failure occurs. The HA feature provides this reset capability through two different mechanisms:

1 VM Monitoring – Guest OS heartbeats issued by the VMware Tools process.

2 Application Monitoring – Heartbeats issued by a program that uses the HA Application Monitoring SDK to communicate with the VMware Tools process and the vSphere HA agent. This mechanism involves local monitoring by the program to avoid the overhead of sending messages to and from vCenter Server.

Figure 3-1 depicts the monitoring and reset capability of host and virtual machine.

Figure 3-1. Heartbeat and Status Signals



Additionally in vSphere 5.5 and later, the in-guest agent can set state to indicate it needs an immediate reset. This can be done without enabling heartbeats. The HA Application monitoring facility can reset the guest OS when ready to do so, if the in-guest agent has not changed state to say reset is no longer needed.

Using the HA Application Monitoring SDK, developers can write HA application monitoring programs in the C or C++ language. The HA Application Monitoring API is available with C language bindings only.

The application monitoring program sends an enable request to start the monitoring, possibly followed by a heartbeat signal. The vSphere infrastructure passes the signal up from your HA application monitoring program to the virtual machine, and then to the ESXi host. The HA application monitoring facility will reset the virtual machine if the application monitoring program stops sending a heartbeat signal, or requests a reset.

For more information about vSphere HA and application monitoring, see the *vSphere Availability* guide in the vSphere Documentation Center.

Prerequisites for HA Application Monitoring

Before you start working with the HA Application Monitoring SDK, make sure that your vSphere application is running within a VMware cluster that has both the **High Availability** and **VM and Application Monitoring** options enabled.

You must install VMware Tools on the virtual machines where your HA monitoring applications are running.

The *vSphere Availability* guide contains information about how to set up a high availability (HA) cluster, and how to configure **VM and Application Monitoring**. VMware's New Cluster Wizard allows you to choose from three monitoring options:

- **Disabled** – Neither VM Monitoring nor Application Monitoring.
- **VM Monitoring Only** – If you choose this option, you will have the Guest OS monitoring discussed previously (the first mechanism).
- **VM and Application Monitoring** – If you choose this option, you will also have the ability to employ Application Monitoring and the HA Application Monitoring SDK (the second mechanism).

For information about Web services interfaces for HA, see the *VMware vSphere API Reference Guide*, especially data objects `VirtualMachineRuntimeInfo` and `VirtualMachineRuntimeInfoDasProtectionState`.

Using the HA Application Monitoring APIs

You can use the HA Application Monitoring SDK to create a stand-alone application monitoring program, or to enhance an existing application or script. The purpose of your application monitoring program determines the API call sequence and the application behavior that you write to handle the response data.

For example, if your application monitoring program is tracking critical applications that are running in a guest OS, your application can intentionally stop sending heartbeat signals if any application-related process fails. The HA monitoring agent interprets the absence of heartbeats as a failure, and resets the virtual machine.

Alternatively, instead of not sending heartbeat signals, your application monitoring program can set the `needReset` flag using the `VMGuestAppMonitor_PostAppState` call. When the HA monitoring agent notices this flag, it will reset the virtual machine.

Most of the calls you make using the HA Application Monitoring APIs send information one-way to the virtual infrastructure of the ESXi host, and the host relays the information to the HA monitoring agent. However the `VMGuestAppMonitor_GetAppStatus` call is a two-way transaction that lets you request the virtual machine status from the HA monitoring agent.

Most HA Application Monitoring functions lack input parameters, because the calls are local. The vSphere infrastructure passes the heartbeat and status data to and from other levels of the cluster.

Call each function from your application monitoring program. The vSphere infrastructure (in the virtual machine where the application monitoring program is running) passes the function data up to the ESXi host. The local virtual machine sends all status responses to your application monitoring program, even though they are passed down from the HA monitoring agent.

HA Application Monitoring API Functions

The following calls are available to a vSphere HA application monitoring program:

Table 3-1. HA Application Monitoring API Calls

Call Name	Data Type Returned	Description
VMGuestAppMonitor_Enable	char	Requests the virtual machine infrastructure to monitor the calling application. The virtual machine infrastructure returns a value of VMGUESTAPPMONITORLIB_ERROR_SUCCESS, if monitoring was enabled. After your application monitoring program makes this call, your program must call VMGuestAppMonitor_MarkActive() at least once every 30 seconds or the virtual machine infrastructure will change the virtual machine's status to Red or Gray.
VMGuestAppMonitor_Disable	int	Requests the virtual machine infrastructure to stop monitoring the calling program. The virtual machine infrastructure returns a value of TRUE, if monitoring was disabled.
VMGuestAppMonitor_IsEnabled	int	Returns the current recorded state of application monitoring. The virtual machine infrastructure returns a value of TRUE, if monitoring is enabled.
VMGuestAppMonitor_MarkActive	char	Sends a request to mark the program as active. This function is also known as the heartbeat because your program must call it at least once every 30 seconds while your application monitoring is enabled, or the virtual machine infrastructure will determine that the monitoring has failed.
VMGuestAppMonitor_PostAppState	int	Publish the application state that the guest OS wants delivered to vSphere HA. The application should monitor its environment and update its state accordingly. Heartbeat counting does not need to be enabled as a pre-condition, so the `enable()` call is not necessary. Returns 0 (zero) on success. The single `state` parameter passed to this call can be either: ■ OK – The guest's application agent declared state to be normal and no action is required. ■ `needReset` – The guest's application agent has requested an immediate reset. The guest can request this at any time.
VMGuestAppMonitor_GetAppStatus	char	Returns the current status recorded by the virtual machine infrastructure as 'Green', 'Red', or 'Gray'. ■ **Green.** Virtual machine infrastructure acknowledges that the application is being monitored. ■ **Red.** Virtual machine infrastructure does not think the application is being monitored. The HA monitoring agent will initialize an asynchronous reset on the virtual machine if the status is Red. ■ **Gray.** Application should send VMGuestAppMonitor_Enable again, followed by VMGuestAppMonitor_MarkActive, because either application monitoring failed, or the virtual machine was vMotioned to a different location. Use the VMGuestAppMonitor_Free function to free the result. If this call returns a nonerror result that you did not anticipate, it may mean that another program in the same virtual machine has called VMGuestAppMonitor_Disable or VMGuestAppMonitor_Enable. If your application is still running, call VMGuestAppMonitor_Enable again, followed by calls to VMGuestAppMonitor_MarkActive.
VMGuestAppMonitor_Free	char	Returns a pointer to the memory to be freed.

Code Sample for appmon.cpp

The HA Application Monitoring SDK includes a code sample called `appmon.cpp`. The sample is located in the `docs/samples` directory and defines the entry point for the console application. The `appmon.cpp` program includes interface code that your application monitoring program can send after receiving results from calls to VMGuestAppMonitor_Enable, VMGuestAppMonitor_MarkActive, and VMGuestAppMonitor_Disable.

Calling the APIs from Your Application

The following steps provide a possible API sequence of calls:

1 Include `vmGuestAppMonitorLib.h` in the declarations for your C program.

2 To start the monitoring, notify the virtual machine that you are going to start sending a heartbeat signal by calling VMGuestAppMonitor_Enable.

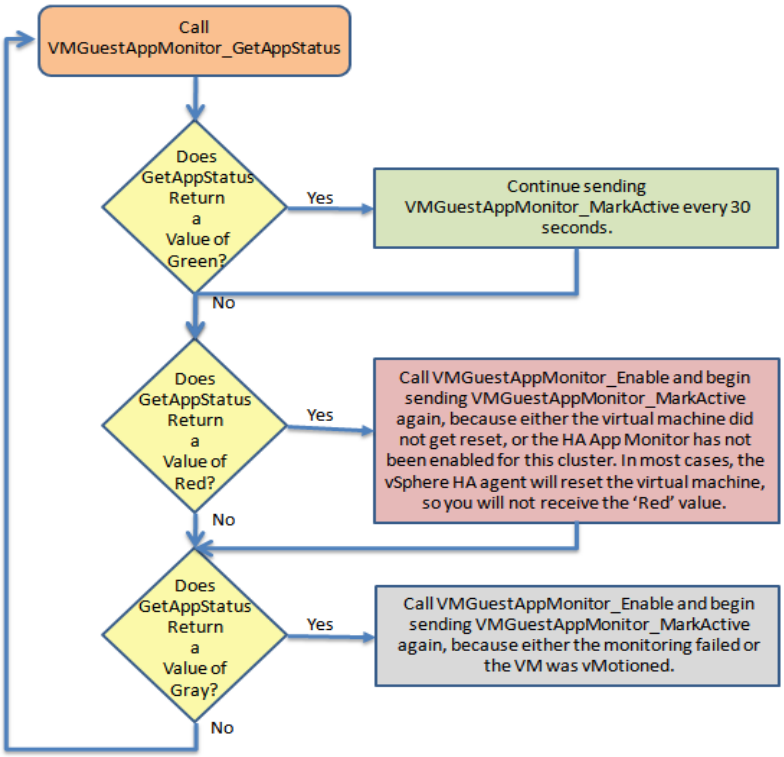
3 After you have called VMGuestAppMonitor_Enable, call VMGuestAppMonitor_MarkActive every 30 seconds or your virtual machine will be reset.

4 Send VMGuestAppMonitor_IsEnabled to make sure the virtual machine infrastructure received your requests correctly and has begun monitoring.

5 Periodically, call VMGuestAppMonitor_GetAppStatus to make sure the vSphere infrastructure is still receiving the heartbeat calls.

The status will be returned as *Green*, *Red*, or *Gray*. See "HA Application Monitoring API Calls," for a description of each status value. Figure 3-1 shows a possible coding flow for the GetAppStatus call.

Figure 3-2. Coding Flow for VMGuestAppMonitor_GetAppStatus



6 After you call VMGuestAppMonitor_GetAppStatus, call the VMGuestAppMonitor_Free function to free the memory that was used to store the status.

If your application does not free the memory, it can use a large amount of storage very quickly, because a new status is created every 30 seconds, when VMGuestAppMonitor_MarkActive is called.

7 Call VMGuestAppMonitor_Disable when you want the agent to stop monitoring.

HA Application Monitoring API Error Messages

The vSphere infrastructure can return errors in Table 3-2 as a result of HA Application Monitoring calls.

Table 3-2. HA Application Monitoring Error Codes

Error Message	Data Type	Code	Description
VMGUESTAPPMONITORLIB_ERROR_SUCCESS	int	0	Call completed successfully.
VMGUESTAPPMONITORLIB_ERROR_OTHER	char		Unknown error.
VMGUESTAPPMONITORLIB_ERROR_NOT_RUNNING_IN_VM	char		Calling application is not running within a virtual machine.
VMGUESTAPPMONITORLIB_ERROR_NOT_ENABLED	char		Monitoring is not enabled.
VMGUESTAPPMONITORLIB_ERROR_NOT_SUPPORTED	char		Monitoring is not supported.

Index