VAssert Programming Guide

Workstation 6.5

This document introduces VAssert, an easy-to-use API that helps you investigate the behavior of applications running on virtual machines, after the applications run. Although VAssert was designed with debugging in mind, you can also use it to study performance and use patterns after the fact.

VAssert allows you to insert replay-only code into your applications, incurring minimal performance overhead at runtime. After you record a virtual machine session, you can replay selected portions while performing data consistency checks and logging. You can ship VAssert code in released software, where it remains inactive and unobtrusive. In case of application failure, users can record a session so you can then activate VAssert code to provide root cause analysis of the problem.

CAUTION This release of the VAssert API is experimental. Interfaces might change in subsequent releases, and backward compatibility is not guaranteed.

About VAssert

In this release, the VAssert SDK is available for Windows guests only. Tested guest operating systems include Windows 2000, Windows XP, and Windows Vista, 32 bit or 64 bit. Windows 2003 Server and Windows 2008 Server are not verified. VAssert supports C++ and C programming.

VAssert uses the VMware® Workstation Record and Replay facility. In the Workstation 6.5 release, you can insert navigation markers while recording or replaying a session and quickly navigate to the markers during replay. You can also browse through a recording and choose the starting point for replay.

In addition, you can insert program statements that initialize the VAssert API, perform consistency checks, and log messages to a file, all at replay time only. The performance penalty of using VAssert is encountered almost entirely at replay time. Replay is deterministic (predictably the same) whether or not VAssert statements are present or enabled. The VAssert SDK is part of VMware Tools.

Example Use Case

The following scenario describes how VAssert could help diagnose the cause of application failure.

1. A customer reports that an application fails, but only after an extended period of heavy use. You cannot reproduce the failure on your development system.
2. You add VAssert checking and logging statements. You recompile the application and deliver it to the customer for test debugging.
3. The customer enables VAssert and begins recording on Monday morning, a time of heavy use. Because VAssert recording is extremely lightweight at runtime, users are unlikely to notice the recording activity. When the application fails, the customer restarts it to avoid service interruption, and turns off recording.
4. The customer sends you Monday's recording. Because VAssert was enabled during recording, checking occurs during replay, and a session log is available. You are able to diagnose the problem. (The recording must come with packaged-up virtual machine, and hardware configuration must be similar.)

VAssert-enabled applications can even run on native hardware (a physical machine without any virtual machines involved). This is not useful for replay debugging, but it is advantageous that the same binary works in both physical and virtual environments.

Traditional Assertions and Logging

Two time-honored methods of debugging are to insert print statements, and to add assertions that verify programming assumptions. The VAssert API provides both methods.

The traditional `assert()` statement is a preprocessor macro defined by including `assert.h` in a C program. If its contained expression evaluates false, `assert()` writes the expression, source filename, and line number to standard error, then calls `abort()` to end the process and possibly produce a memory image. If disabled by `NDEBUG`, `assert()` has no effect.

The `VAssert_Assert()` statement is similar, but takes effect deterministically at replay time, unless you specify otherwise. Like traditional `assert()`, `VAssert_Assert()` is a macro.

Most system software sends activity and security-related messages to a log file in a well-known location, and application software often does the same. The `VAssert_Log()` statement sends messages to the `vlog.txt` file in the virtual machine's directory at replay time, unless you specify otherwise. `VAssert_Log()` is also a macro.

NOTE VAssert statements must be enabled at record time. See "Enabling VAssert Recording" on page 3.

Best Practices

Although VAssert was designed for debugging, you can also use it for testing and performance analysis.

Offline Testing

You can use VAssert to improve the efficiency of automated offline testing. Normally, checking and logging I/O alters application behavior because of performance overhead. But with VAssert, software developers can include elaborate assertion checking and extensive logging in their code, and in-house QA engineers can record testing sessions without suffering from slow performance. The recorded sessions can be replayed in the evening when there is less traffic, with expensive checking and logging activated. Application replay can also be completely automated. Later, developers can investigate and debug failures, if any.

Performance Analysis

To analyze performance of a database buffer pool, you can add VAssert code to collect a history of database accesses. This logging overhead normally skews the results if performed live online. With VAssert, you collect the log afterwards, without disturbing the access history. You can then analyze the data to design a better buffer pool replacement algorithm.

Setting up VAssert

This section describes how to install, locate, and enable VAssert.

Upgrading VMware Tools

When you install VMware Workstation, it creates a VMware Tools directory on the host, which contains an upgrader application to create a complete VMware Tools package on guest virtual machines.

To install the VAssert SDK

1. After installing a Windows guest operating system, on the guest select **VM > Install VMware Tools**.
2. Follow documented procedures to install or upgrade VMware Tools.

   On Windows guests, the default VMware Tools folder is `C:\Program Files\VMware\VMware Tools`.

Locating the VAssert SDK

The VAssert package is now installed on the guest in the folder `VMware\VMware Tools\VAssert SDK`, which contains the following files:

- `bin\win*\libVAssert.lib` – Link-time library for VAssert programs.
- `lib\win*\libVAssert.dll` – Dynamic link library for VAssert runtime.
- `examples\libVAssertTest.c` – Sample application with command-line options for VAssert statements.
- `include\vassert.h` – Header include file for VAssert programs.
- `include\vassertShared.h` – Internal file used by `vassert.h`; not important for programmers.

The VAssert SDK is tested with Microsoft Visual C++ 6.0, although later versions of Visual C++ are known to work. You need the SDK to modify your programs for debugging with VAssert. Add the header files to your project, and link with the `libVassert.lib` file.

Enabling VAssert Recording

VAssert functionality is disabled by default, because it incurs a small amount of overhead in the guest operating system, whether or not applications are using VAssert features.

To enable VAssert

1. Inside the Workstation host, shut down and power off the Windows guest.
2. In the Workstation user interface, select **VM > Settings > Options > Snapshot/Replay**.
3. Under Replay, select the **Enable VAssert** check box and click **OK**.

After enabling, any recorded application that includes VAssert can execute replay-time code when replayed, regardless of the check box setting at the time of replay. Recordings created when the check box was deselected can never run replay-time code, whether or not the application includes VAssert.

Enabling VAssert adds the following settings to the guest virtual machine `*.vmx` file. The first setting turns on VAssert features, and the second setting permits internal communication.

```
vassert.enabled = "TRUE"
isolation.tools.vassert.disable = "FALSE"
```

One related setting is not check box enabled in this release. It permits the use of VAssert statements to start and stop recording. See "Using VAssert_StartRecording()" on page 4 and "Using VAssert_StopRecording()" on page 5. You might want to add it to the `*.vmx` file.

```
isolation.tools.stateLoggerControl.disable = "FALSE"
```

Programming VAssert

This section describes how to call statements in the VAssert API.

Initialization

To initialize VAssert, VMware Tools must be installed on the Windows guest. Add the `vassert.h` header file to your C or C++ program, so `VAssert_Init()` can load the DLL and initialize the library:

```
#include "vassert.h"
...
    if (!VAssert_Init()) {
        cout << "Warning: cannot initialize VAssert library." << endl;
        goto continue_without_vassert;
    }
```

This function fails if `isolation.tools.vassert.disable` is set to TRUE. If you want to leave VAsserts statements in production programs and enable VAssert without recompiling, your application can ignore the return value of `VAssert_Init()`.

Using VAssert_StartRecording()

You can use the Record and Stop facility in the Workstation user interface to record all or part of a VAssert-enabled application running on a guest virtual machine. However, you can get finer-grained control by placing a `VAssert_StartRecording()` statement exactly where you want it in your program.

```
if (!VAssert_StartRecording()) {
    cout << "Warning: cannot start VAssert recording." << endl;
}
```

A virtual machine must be powered on to start recording. You cannot begin recording when a recording is already underway, nor can you begin recording during replay.

You probably want to match `VAssert_StartRecording()` with a `VAssert_StopRecording()` statement.

Using VAssert_Assert()

The `VAssert_Assert()` statement is similar to other `assert()` statements, but takes effect deterministically at replay time, unless you specify otherwise. If VAssert was enabled during recording, at replay time the `VAssert_Assert()` macro evaluates its expression and terminates the application if the expression is false.

For example, before calling functions that expect a pointer to data and might crash given a null pointer, you can assert that the pointer is non-null:

```
VAssert_Assert(ptr != 0);
```

You can also check for memory problems inside a long-running program as described in "Example Use Case" on page 1 by asserting that memory allocation is within range after `malloc()`:

```
VAssert_Assert(((char*)p) >= min_address);
VAssert_Assert(((char*)p) <= max_address);
```

In some cases, you might want to replace the underlying `abort()` routine so that the program does not fail, but instead provides helpful diagnostic information. See the define option `VASSERT_CUSTOM_ABORT` in "Compiler Flags" on page 7.

Going Live During Replay

The Record and Replay facility allows you to interrupt an in-progress replay at any time and begin interacting with it. You can control this with the **Go Live** button in the user interface or programmatically with VAssert. At the point in your code where you want the application to go live, insert the following statement:

```
VAssert_Assert(FALSE);
```

The `vassert.h` include file defines `VAssert_GoLiveMain()`, which `VAssert_Assert()` calls internally.

Returning to Replay

You can return to replay by inserting the `VAssert_ReturnToReplayMain()` statement inside a VAssert code block. Make sure that the application is in replay state; otherwise this statement might cause the application to crash. Your program does not have to clean up program state, because VAssert restores it before returning to replay, thereby eliminating side effects from the `VAssert_Assert()` code block.

```
if (vassert_state.inReplay)
    VAssert_ReturnToReplayMain();
```

One possible use for this statement is to exit early from a VAssert block. For example, in the code sequence below, `VAssert_ReturnToReplayMain()` exits both `firstFunction()` and `secondFunction()` without returning back through the calling code path.

```
VAssert_Assert(firstFunction()); // calls
→ firstFunction(); // which calls
    → secondFunction(); // which calls
        → VAssert_ReturnToReplayMain(); // end current VAssert and resume replaying
```

This is for advanced programmers.

Using VAssert_Log()

If VAssert was enabled during recording, at replay time `VAssert_Log()` sends `printf()` style output to the `vlog.txt` file on the VMware host for its respective virtual machine. For instance, a Windows XP guest on a Windows host sends output to `C:\My Documents\My Virtual Machines\WindowsXP\vlog.txt` or some similar path name. On Windows, `VAssert_Log()` calls `_vsnprintf()` internally, which is slightly different from the POSIX standard `printf()`. See the MSDN for reference information:

http://msdn2.microsoft.com/en-us/library/56e442dc(VS.71).aspx

Because `VAssert_Log()` is implemented as a preprocessor macro, you must place the `printf()` format string inside double parentheses:

```
VAssert_Log(("Customer transaction number %u at %s\n", transactNum, ctime(&time) ));
```

This logging statement is useful for instrumenting your application without causing it to exit. For example, the long-running program in "Example Use Case" on page 1 could write a log message before every operation with high overhead, giving you an idea of where it failed.

The `VAssert_Log()` statement might fail when printing floating point numbers expressed in `%f` or `%g` format.

Using VAssert_StopRecording()

If you are interested in only a certain section of your application, you can stop a recording in progress with the `VAssert_StopRecording()` statement. Otherwise, recording continues to the end of program execution.

```
if (!VAssert_StartRecording()) {
    cout << "Warning: cannot stop VAssert recording." << endl;
}
```

VAssert Reference

This reference includes sections about API statements, configuration options, compiler flags, and limitations.

API Statements

This release has five documented APIs:

Initialize

The initialization function returns `TRUE` if it is able to initialize the VAssert library; otherwise it returns `FALSE`.

```
#include "vassert.h"
extern char VAssert_Init(void);
```

Record

This statement begins the underlying system recording facility, sending to a snapshot object.

```
#define VAssert_StartRecording() VAssert_SetRecordingMain(1)
```

This function is effective only when the **Enable VAssert** check box is selected in the Workstation user interface.

Assert

If `<expression>` evaluates `FALSE`, this statement calls the system `abort()` routine to terminate the process, which you can attach with a debugger to determine the filename and line number of the failing assertion.

```
#define VAssert_Assert(<expression>) ...
```

It is possible to replace both the `assert()` routine and the `abort()` routine it calls; see "Compiler Flags" on page 7. You could for instance replace the `abort()` routine with one that calls another routine to emit `__FILE__` and `__LINE__` variables from the preprocessor.

Log

The log statement sends formatted output to the `vlog.txt` file of its respective virtual machine on the host.

```
#define VAssert_Log((<printf-string>)) ...
```

The `VAssert_Log()` statement might fail when printing floating point numbers expressed in `%f` or `%g` format.

Stop

This statement ends the underlying system recording facility, closing the snapshot object.

```
#define VAssert_StopRecording() VAssert_SetRecordingMain(0)
```

This function is effective only when the **Enable VAssert** check box is selected in the Workstation user interface.

Configuration Options

Table 1 shows configuration options supported in the `*.vmx` file to control the behavior of VAssert. Options to control the logging behavior of `VAssert_Log()` are especially useful for customizing replay.

Table 1. VAssert Configuration Options

Option Name	Settings	Default	Description
`vassert.enabled`	TRUE FALSE	FALSE	Determines whether VAssert functionality is enabled for new recordings. The default FALSE means that VAssert is not enabled.
`isolation.tools.vassert.disable`	TRUE FALSE	TRUE	Determines whether VAssert functionality is exposed to guests. Use this only in conjunction with `vassert.enabled`. The default TRUE means VAssert functionality is not exposed to guests.
`vassert.logFile`	<filename>	vlog.txt	Destination filename for `VAssert_Log()` output, created in the guest virtual machine's directory on the host.

`vassert.logMode`	`"prompt"`
	`"append"`
	`"replace"`
	`"discard"`
		`"prompt"`	Indicates what to do when a `VAssert_Log()` produces output, and the log file already exists.
			"append" means append to the log file.
			"replace" means replace the log file.
			"discard" means preserve the old log file and discard any new log output.
			"prompt" means to prompt with a dialog box, so the user can select one of the above options.

`vassert.maxLogFileSize`	<number>	–1	Limits the log output file to a maximum file size in bytes. The default –1 means no maximum limit, so the log file can grow as much as needed.
`vassert.maxTime`	<number>	10	Limits the amount of time spent executing `VAssert_Assert()` to the given number of seconds. Any invocation lasting longer than that is terminated. Specifying zero means that no limit is enforced. The imposed limit is not exact. It is never less than the specified duration, although it could be more.

Compiler Flags

In the SDK, compiling with the `VASSERT_ALWAYS_EXECUTE` flag causes the `VAssert_Assert()` and `VAssert_Log()` calls to be treated as normal assert and log statements at runtime. In other words, they execute all the time, like standard `assert()`, not only at replay time.

You can define the following macros before including the `vassert.h` header file in a program. These macros specify replacement abort, assert, and log statements:

- `VASSERT_CUSTOM_ABORT` — When an assertion fails, VAssert ends replay and makes the virtual machine go live, so it begins accepting new input from the user. If this occurs, `VAssert_Assert()` inserts the `VASSERT_CUSTOM_ABORT` macro to halt the application and report an error. The `VASSERT_CUSTOM_ABORT` macro by default calls the standard library `abort()`, which prints a generic error message and allows you to attach a debugger to get details about the failure. You can alter this macro to provide more information, such as the `__FILE__` and `__LINE__` number of the failed assertion. Because `VASSERT_CUSTOM_ABORT` is inserted after the end of replay, the I/O restrictions in "Limitations" on page 7 do not apply. This macro can print output.
- `VASSERT_CUSTOM_ASSERT` – If the `VASSERT_ALWAYS_EXECUTE` flag is defined at compile time, `VAssert_Assert()` becomes a wrapper around `VASSERT_CUSTOM_ASSERT`, whose default definition is to call the standard library `assert()`.
- `VASSERT_CUSTOM_LOG` – If the `VASSERT_ALWAYS_EXECUTE` flag is defined at compile time, `VAssert_Log()` becomes a wrapper around `VASSERT_CUSTOM_LOG`, whose default definition is to call the standard library `printf()`.

Limitations

Replay-time code running with VAssert has the following limitations:

- Vassert code cannot perform I/O or cause I/O to be performed. It cannot execute `in` and `out` assembly instructions, nor can it read or write memory-mapped regions. Swapping might cause implicit I/O, so do not run VAssert code near physical memory limits. When the VAssert library detects I/O, it terminates and skips the offending replay-time code. Subsequent VAssert execution remains undisturbed.

  You cannot display output using `VAssert_Assert()`, because calling `printf()` or opening a dialog box inherently produces I/O. The only way to produce user-visible output is by using `VAssert_Log()` or by defining `VASSERT_CUSTOM_ABORT`, which is allowed to produce I/O because it executes after an assertion has failed and replay has stopped.
- Replay-time code cannot depend on the delivery of interrupts. For example, timer interrupts and I/O interrupts are not delivered during replay-time execution.
- If a guest application terminates abruptly, the VAssert DLL might de-initialize improperly. This could happen when you forcibly terminate processes with Windows Task Manager. If improper de-initialization occurs, spurious VAssert invocations might cause replay glitches. At worst, spurious invocations can cause replay to halt indefinitely, although you can manually stop replaying if you notice that progress has terminated. Normally, spurious VAssert invocations have no noticeable effect during replay, and valid VAssert invocations run undisturbed. When the VAssert DLL re-initializes because of a `VAssert_Init()` call in the recording, it prevents spurious invocations during further replay.
- Replay-time code must not modify the privileged guest CPU state. This includes CR0, CR4, and Machine Specific Registers. However, modifications to normal non-privileged CPU state, such as the CPU general purpose registers and flags, the segment registers, and address space modifications (that is, assignments to CR3) are permitted.
- When replaying a recording that has VAssert enabled, the **Suspend** and **Add Marker** operations are disabled in the Workstation user interface.

Performance Implications

The VAssert APIs defer execution of some code from recording to replay. This generally makes the record-time execution of a program faster when VAssert calls are not evaluated, and the replay-time execution slower when VAssert calls are evaluated. To implement this facility, the VAssert library must add overhead to the replay-time execution. Testing indicates overheads of hundreds of microseconds per VAssert. Depending on the rate of VAssert invocations per second, this can make replay-time execution much slower than record-time execution. For example, if during recording, your program runs 15 million VAssert invocations in one second, replay-time execution could take substantially longer (15 million x 500 microseconds = over 2 hours).

Plan for these performance slow-downs either by using more complex but more sparsely packed VAsserts, or by scheduling an appropriate amount of time to perform replay, such as overnight or batched.

Support and Feedback

See the “Record and Replay” chapter in the *Workstation 6.5 User’s Manual* for information about the underlying facility and its user interface.

If you encounter any issues while programming VAssert, post your questions in the VMware Workstation 6.5 communities forum or in VMware communities that discuss developer products. If you have any technical inquiries or suggestions, send email to vassert-support@vmware.com.