

# Resolve Memory Leaks Faster

Run The UMDH Tool To Get Key Problem-Solving Data And Cut Support-Call Time  
Michael Morales

(Reprinted From WindowsItPro Magazine)

## Executive Summary

You probably can't avoid tech support problems entirely, but by using tools that Microsoft's Global Escalation Services support team uses, you can obtain detailed Windows system and application information that can shorten a support call or even avoid it. In this first What Would Microsoft Support Do? column, Microsoft Escalation Engineer Michael Morales shows you how to use the user-mode dump heap (UMDH) tool (umdh.exe, one of the Debugging Tools for Windows) to diagnose a process memory leak on a system and use UMDH's output to solve the problem more quickly.

If you manage a Windows environment, you know that a call to tech support is an inevitable part of your job. But there are things you can do to help resolve support issues faster—and perhaps avoid the dreaded support call entirely. In nine years as an escalation engineer for Microsoft's Global Escalation Services support team, I've found a number of Microsoft tools especially helpful for resolving tech support issues. In this new column, What Would Microsoft Support Do?, I'll show you how you can use these tools to obtain valuable information that will help you either facilitate your tech support call or research your own solution. We'll start our exploration of Microsoft's troubleshooting tools by walking through using the user-mode dump heap (UMDH) tool to identify and solve a memory-leak problem.

## Troubleshooting a Memory Leak

UMDH (umdh.exe) is part of the Debugging Tools for Windows. UMDH aids in troubleshooting process memory leaks by revealing the components responsible for allocating the most memory. You can use UMDH with Windows Server 2008, Windows Server 2003, Windows 2000 Server, Windows Vista, and Windows XP systems.

I recently used UMDH to solve a customer's memory-leak problem. The customer's Performance Monitor logs indicated that the svchost.exe process was leaking enough memory to cause the entire system to crawl. However, the information didn't pinpoint what components were involved in the leak or the functions those components executed—information that UMDH could provide.

## UMDH Steps

Using UMDH to troubleshoot a memory leak involves a sequence of straightforward steps. Here's the process:

1. Use the gflags.exe tool to enable the registry setting Create user mode stack trace database. This setting lets the system store the process's function calls and module listing in a database during execution; UMDH then dumps the database into an output file. Gflags is installed when you install the Debugging Tools for Windows. This sample gflags.exe command enables a setting for the notepad.exe process:

```
gflags.exe -i notepad.exe +ust
```

The command sets a registry value that's read by the system during process startup and lets the system keep track of the threads that allocate memory inside the process. After running gflags.exe to enable the setting, you'll need to restart the process before you can perform step 3. Also,

## Resolve Memory Leaks Faster

Run The UMDH Tool To Get Key Problem-Solving Data And Cut Support-Call Time  
Michael Morales

(Reprinted From WindowsItPro Magazine)

remember to turn off the setting after you've completed the necessary tracing for a leaking process. The following command disables gflags.exe for the notepad.exe process:

```
gflags.exe -i notepad -ust
```

2. Set up the Microsoft symbol path to point to the Internet for symbols. Enabling symbols lets UMDH output the process trace information in a readable format. Without symbols, each line in the trace output will show the word "module" instead of an actual .dll name and numbers instead of the function name (more about the trace output shortly).

To enable symbols, right-click My Computer, click Properties and the Advanced Tab, then click the Environment Variables button. Under System Variables, click the New button, and in the Variable name box, enter

\_NT\_SYMBOL\_PATH

In the Variable value box, enter the symbol path srv\*c:\symbols\*. UMDH will use the symbol path to display the components responsible for leaking memory. (This symbol path is valid for Server 2008, Windows 2003, Win2K, Vista, and XP.)

3. Now you can take your first UMDH snapshot. To do so, from the command line, navigate to the location where you've installed the debugging tools. Then enter a command like this:

```
C : \debug>umdh -p:260 -f:Notepad1.txt
```

(Here, I installed the tools in the C:\debug directory.) The -p: is the process ID of the leaking process (which you can obtain from Performance Monitor or Task Manager), and the -f: is the name you've chosen for the first snapshot file.

4. Allow enough time between the first and second snapshots to ensure that the process leaks memory. While you're waiting between snapshots, you can use Performance Monitor to see how much memory is being leaked.
5. Take your second snapshot, for example

```
C : \debug>umdh -p:260 -f:Notepad2.txt
```

6. Now compare the two snapshots, by running a command like this:

```
: \debug>umdh -v Notepad1.txt  
Notepad2.txt >c:\comparefiles.txt
```

The -v parameter tells UMDH to include in its output summary information that describes how much memory each thread has consumed between the first and second snapshots (more about

## Resolve Memory Leaks Faster

Run The UMDH Tool To Get Key Problem-Solving Data And Cut Support-Call Time  
Michael Morales

(Reprinted From WindowsItPro Magazine)

threads shortly). You need to specify a file to contain the output for the snapshot comparison; here, the filename is comparefiles.txt.

The previous command's output lists the components and function calls that allocated the most memory within the process. Having this detailed information about the process will make the problem easier for tech support to pinpoint and resolve— or will give your systems administrator adequately specific information to research the problem further and possibly update the binaries involved in the leak.

A note about using UMDH: You can trace both Microsoft and non-Microsoft related processes and services by running UMDH commands; however, to actually capture the component name involved in the leak, you'll need the corresponding symbol file for that component. Some vendors don't make their symbol files public; if you don't have access to the symbol file, the information in the UMDH output file will be limited to only the component's load address and exclude the component name and function being executed. So, to get any meaningful output from UMDH, you should specify at least the Microsoft symbol path, as explained earlier.

### Interpreting UMDH Output

When you open the output file, comparefiles.txt in Figure 1, at the top you'll see the first thread of execution (thread for short), two lines followed by a succession of lines grouped together. Threads represent a running task inside a process; they're components and functions that have memory allocated. Every process must have at least one thread to be able to load and run. The top two lines are the thread's summary information, and the group of lines under the summary information represents each thread entry in that process. Let's look more closely at the output and what it means.

The first two lines of the thread stack show comparative memory-usage information from the two snapshots. The first hexadecimal number, +113faf000, represents the delta change in memory consumption from the first snapshot to the second. So in our example, you can see a change of more than 4.6GB of memory ( $113faf000 - 0 = 4.6\text{GB}$ ). To see the delta value, you can convert the hex value to decimal by using the Windows calculator's Scientific view (you can access the calculator either through the Start menu or by running calc.exe).

The next number, 81df8, represents the number of actual allocations that occurred to consume the memory. 81df8 hex represents 531,960 allocations. This high number of allocations is normal, considering this thread is responsible for more than 4GB of memory. The next part, BackTrace8117, is the internal ID with which the system has tagged this thread.

The thread at the top of the UMDH output is the thread that consumed the greatest amount of memory, so that's where you'll start investigating the memory-leak problem. Each thread in the output file consists of the component's load address (e.g., in the first entry, 77EDCA76), the component filename (or DLL name—ntdll in the first entry), and just after the ! sign, the function within the DLL that was executed (e.g., !\$\$VProc\_ImageExportDirectory).

## Resolve Memory Leaks Faster

Run The UMDH Tool To Get Key Problem-Solving Data And Cut Support-Call Time  
Michael Morales

(Reprinted From WindowsItPro Magazine)

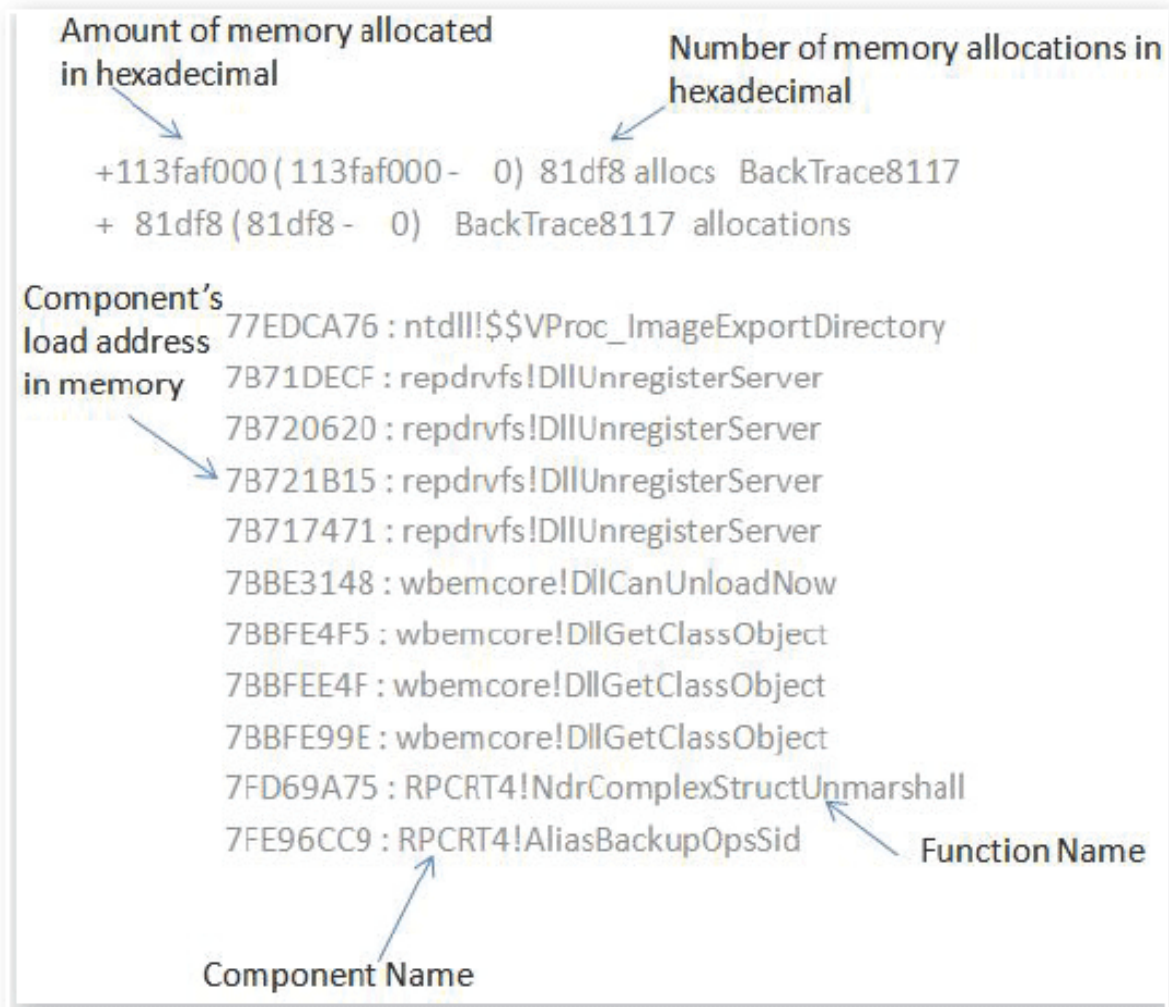


Figure 1: UMDH output showing topmost thread

You can use the UMDH output to start your troubleshooting investigation by reviewing the components in the output file and, if necessary, updating them to their latest versions. If the components involved in the process are up-to-date and the leak still occurs, your next step is to call tech support or research the problem further.

### Using UMDH Information

You can further narrow down and possibly solve the problem by researching it online. For example, I searched on information from the sample UMDH snapshots—the string “repdrvfs wmi leak,” including “wmi” because the leak occurred in a Windows Management Instrumentation (WMI) process and

## **Resolve Memory Leaks Faster**

**Run The UMDH Tool To Get Key Problem-Solving Data And Cut Support-Call Time  
Michael Morales**

**(Reprinted From WindowsItPro Magazine)**

“repdrvfs” because that component name was high on the thread stack (i.e., the thread that was consuming the most memory) and repeated several times (indicating that the repdrvfs DLL was involved in the consumption of memory). My search found a TechNet article that provided the fix for the problem, at [support.microsoft.com/kb/838884](http://support.microsoft.com/kb/838884). Thus, when you select components to search, you’ll probably be most successful searching those that are both high on the thread stack and repeated.

Of course, you won’t solve all leaky-application problems by using UMDH. However, using UMDH for troubleshooting leaky processes will provide key information that can significantly reduce the time needed to resolve a technical support issue. Check out the Microsoft Advanced Windows Debugging and Troubleshooting blog ([blogs.msdn.com/ntdebugging](http://blogs.msdn.com/ntdebugging)) for further guidance in identifying and resolving Windows technical issues.