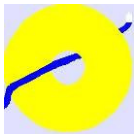


TMurgent Technologies
26 Angela Street
Canton, MA 02021 USA
(+1) 781.492.0403

When Applications Crash
Part I - Watson

A two part guide to how any ISV can find and fix crashes that occur at customer sites.

Tim Mangan
Founder, *TMurgent Technologies*
April 1, 2006

Introduction

Unfortunately it is inevitable. No matter how careful your developers are. No matter how thorough your testing is. It will happen. At a customer site. Before they buy your software.

Something crashes in your software. Buffer overrun. Dereferencing a null pointer. Divide by zero. You name it, it can happen – and will at some point.

This white paper, and its companion[1], covers the basics of two methods you can use to solve these problems when they happen. Both require some advance planning on your part.

The first method, covered in this paper, is an old tried-and-true method that has been around for years. The second is newer and usually easier for your customer.

Method 1 – "'Mr. Watson--come here--I want to see you."[2]

Whether it was named for assistant used by Alexander Graham Bell, or that of the famous fictional detective Holmes, "Dr Watson" has been around helping software vendors with their windows applications for some time.

drwtsn32.exe is an OS component shipped with retail versions of windows. You can find it in the system32 folder of windows. Watson configures the system to capture information any executable that has an unhandled exception. The capture will consist of two files, a log file and a binary file of the offensive process.

Technically, Watson is a debugger. When the OS is installed, it is registered as the debugger of choice via the registry key
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\AeDebug
If you installed another debugger on the target system this can be overwritten. Starting Watson with the "-i" parameter ("drstsn32.exe –i") will reset it.

Setting up Watson

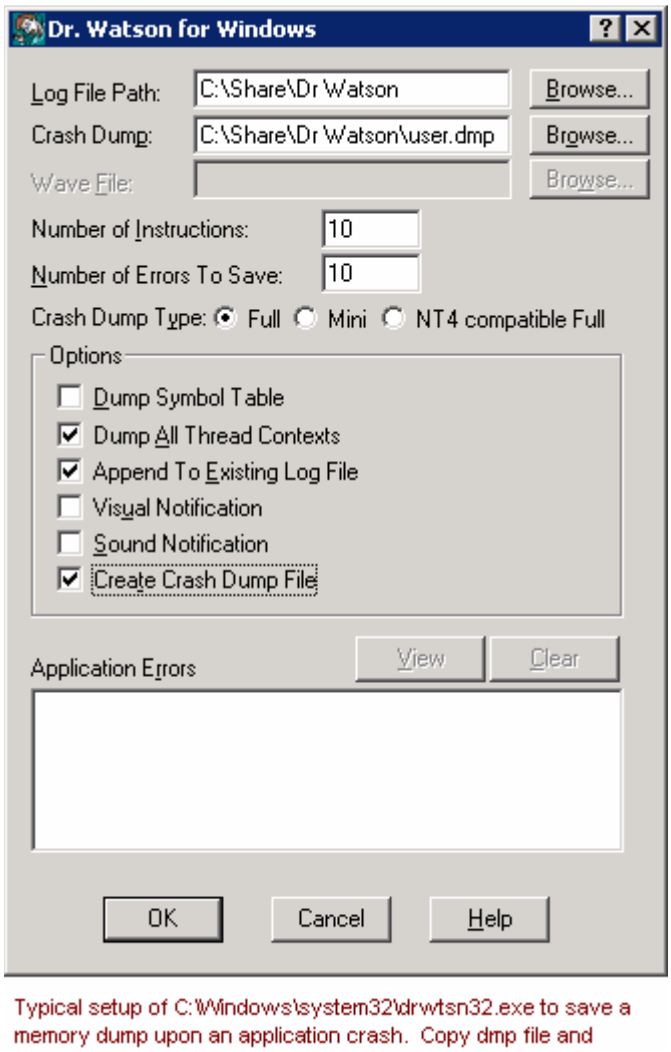
Information on drwtsn32 is available from Microsoft. The Knowledge Base article **kb308538**[3] is a good place to start. drwtsn32.exe is the gui based tool used to configure your setup. Figure 1 shows this GUI with a typical setup.

---

[1] See *"When Applications Crash, Part II"* www.tmurgent.com/WhitePapers/WhenAppicationsCrashP2.pdf *April 2006.*
[2] Alexander Graham Bell – Notebook "Experiments Made by A. Graham Bell", pages 40-41. 1876. Library of Congress.
[3] See http://support.microsoft.com/default.aspx?scid=kb;EN-US;308538

You can select whatever folders you like, but keeping the log file and dmp file in the same folder probably makes sense. The user running the faulting application needs write permission to this folder and these files.

The Number of Instructions can usually be ignored if you are saving the crash in a dmp file. This affects information that will be in the log file. For each thread in the faulting process, this number affects the number of CPU instructions around the faulting instruction that will be written to the log file.



Typical setup of C:\Windows\system32\drwtsn32.exe to save a memory dump upon an application crash. Copy dmp file and

Figure 1 - dwstsn32.exe

The Number of Errors to Save field is important. The system keeps a registry entry **HKEY_LOCAL_MACHINE\Software\Microsoft\DrWatson\NumberOfCrashes** that records the number of process crashes. Although the visible registry settings are also stored under the DrWatson registry key, there is no GUI access to the NumberOfCrashes

key value. So after a number of crashes, (which seems to be one more than the limit you set in the GUI) Watson will simply stop recording additional crashes. You must reset this NumberOfCrashes field using the registry editor when this happens.

Also of importance is the the Log Path and Dump file location. These fields default to the user space (C:\Documents and Settings\username\...) of the user logged in running this GUI – but are stored as relative to the user environment variables. When a crash occurs the debugger will interpret any such folders based on the context of the crashed application.

You really want to create a special folder that is user independent for this operation. That folder must have read-write permissions available for everyone. For example, if you are working with a system service running in the local system context that is crashing and you left these folders pointing to your home folder, the crash might end up in the "default user" folder! So creating a special folder so that you know where the dumps will end up will save you a few missteps in locating the dumps.

The full crash dump type will capture the entire process memory space, and although it may be large (depending on the application), will ensure that you can debug the problem.

The options selected in the figure are typical. If you include symbols in the application (which typically is not done in release builds of software) checking the dump symbol table can be very handy. Leaving the sound and visual notifications is usually preferable on production systems, as we don't really want the end-user being involved. This is also the appropriate choice for windows services.

Once configured, click OK and the system will do the reset. When an unhandled application exception occurs, Watson will start up as a debugger, capture the image as requested, and report the event in the Windows Event Log. You can monitor the event log or just look for the dmp file to appear to know that an exception has been trapped.

When you have captured the event, compress the log and dmp file and send them back.

The Watson Log File

The log file is a text file that contains a number of sections. If you selected the "Append to existing log file" option, you will need to scroll down to the last exception in the log to find the one related to the current dmp file. Each exception contains the following sections:

- The application, its process id, time of fault and an exception number.
- Next is information about the system, then a list of all processes that were running, including the faulting app, at the time of the fault. (The task list for the first exception in that log has some errors in it which I have never seen happen before, but the others are OK).
- Next is the module list, all exe and dll files in use by the faulting application.
- Finally we get a state dump for each thread in the faulting process.

If you are taking a crash dump image also, the log file mostly allows you to confirm that the dmp is from the application crash you intended to capture. It can be some help to see system and application version information, as well as what other processes were running in the system at the time.

The DMP file

The dmp file is a memory image of the process at the time of the exception. You can load this into Visual Studio (or windbg, or kdbg, etc…) by double clicking on the dmp file on your development station. Click Run and the same CPU instruction will be attempted again causing the exception to appear.

Since this is probably from an exe that was stripped of symbols, you will want to bring in a symbol file from the same build. This is where you needed to think ahead. Your build process should always save the pdb files from release builds. Having the exact source code version on your development system is important also – so tagging in your source control system is important too. If you didn't think ahead, you may need to rebuild and ship to the customer for a second try at this. But go ahead and try to debug anyway – the problem might be painfully obvious.

If the dmp file is a mini-dump, then the amount of information in the dump is limited. This means you have limited stack, memory, and code visibility. Full dumps give you the entire picture and are preferred.

I once had to find a problem by running the mini-dump file debugging in one window and a loaded version of the executable in another and locating the faulty line of code by matching up the disassembled code, so that is possible in a pinch.

Looking Ahead

In Part II of this paper, we look at using the Windows Error Reporting (WER) Service and Winqual as an alternative to Watson.