

DebugLive: A New Approach to Debugging

Author: John Robbins

In today's age of collaboration, there are twelve year olds working on their homework assignments across continents through the Internet. In the business world, we are collaborating through tools like SharePoint, internal blogs, or to the constant worry of CIO's everywhere, public tools such as Google Docs or Twitter. The one area we are not collaborating is when doing the hardest part of software development and testing: debugging.

What usually happens is that a customer says they have experienced a problem and the bug report bounces around through various levels and finally ends up on a developer's desk. The developer starts digging in and invariably becomes stuck because there's either not enough information about the problem, the dreaded "no repro" problem, or they just don't know where to turn next. If they are lucky, they can call in a co-worker to sit at the computer with them and the two can engage in a bit of "Extreme Debugging" where one drives the computer and the other looks over their shoulder and offers ideas and analysis.

In a perfect world, in any debugging challenge the right developer you need to assist is always in the office and has the time to help. As we all know, the reality is never so nice. The person stuck with solving the bug could be a support person who doesn't have enough knowledge of the code base to understand anything. The developer in the company that can help the most might be in the office on the other side of the world. The best case is that the bug takes weeks to fix and costs hundreds of thousands of dollars. The worst case is that the bug doesn't get fixed and the company goes out of business. What we need is a tool that will make collaborating on bugs as easy as sharing information on Facebook.

As someone who has debugged applications for many companies around the world the biggest problem, a major issue we face is actually getting a debugger attached to the process. Because of security limitations, VPN problems, or poor infrastructure on the client's part, about the only way to get that debugger attached is to physically travel to the company, walk into the server room, and manually attach the debugger. Obviously, this is not very cost or time effective. If my client could just say something like "here's a secure connection to our debugging service, can you take a look?" that would save everyone a ton of hassle and my clients a ton of money in travel costs.

A secure collaboration service for debugging is a primary goal of DebugLive. DebugLive wants to be the debugging service and repository for any debugging information at your company. The idea is that when anyone in your company needs debugging help there is a central location where everything is consolidate. It instantly becomes easier to deal with problems no matter where or who is looking at them. More importantly, it allows you to tap the exact expert you need to look at the bug so you can get the problem solved as fast as possible.

You're probably thinking that getting DebugLive set up for your environment is going to be a hassle. When something promises to be your debugging repository with debuggers and other tools, you'd

expect that it would involve all sorts of installation and policy management to deploy and use. Any time the word “install” is used around network administrators, they immediately go into that massive defensive mode where the only words out of their mouth are “no” with possibly a mean adjective in front of that “no.” This is especially true of those times when you say you need to install something on a production server. That generally throws a network administrator into spasms of no-ness.

Since most debugging challenges occur on production machines, the team at DebugLive realized the need to combat the refusal of network administrators to install anything at all. To get the DebugLive tools on your machine requires a few simple steps. The first is to open a web browser, connect to www.DebugLive.com and log in with your account. Step 2 is very hard: download the debugging component by clicking the Start Debugger or Start Remote Debugger link. The final step is simply running the downloaded DebugLive Local Starter application. If you’ve already downloaded the DebugLive Local Starter application onto one machine, you can copy it over to other machines. Having fought hard against the network administrator “no” many times before, I’m thrilled that DebugLive makes it super simple to get the debugging components down to a machine. While you may think I’m touting a small feature, but in production environments no install is a huge feature!

While DebugLive is a debugging environment, which I will talk about later in this whitepaper, it’s certainly not limited to just developers. One of the most important constituencies for DebugLive are testers. In many development shops, testers generally do not have a technical background. Their job is to perform repetitive tasks to validate functionality and try to expose problems. In those situations, when those testers run into issues, such as crashes or memory leaks, they report that the test didn’t go as expected. At that time, they have no real information to give the developer to track down the root cause.

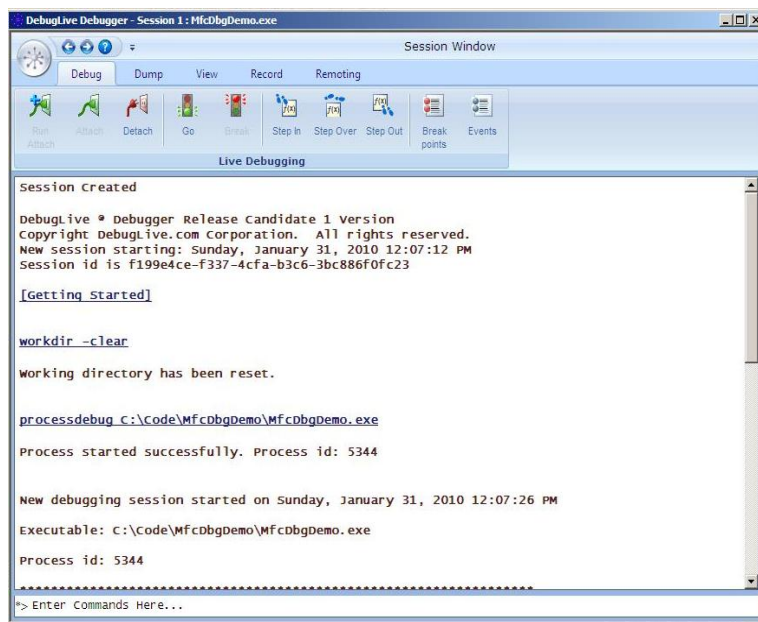
With DebugLive, it’s trivial for a tester to create a new debugging session and either start the process under the debugger or attach to an already running instance. As they work through the manual test process and they encounter a problem they can instantly create a remote debugging session. Now they can contact the developer, report the problem, and give the developer control with the process at the exact location where the problem occurred. Instead of requiring the tester to learn difficult tools like the Debugging Tools for Windows (WinDBG), the tester can get up to speed on DebugLive in five to ten minutes of training. The easier it is for a tester to give the developer the application in the exact problem state, the easier it is for the developer to debug and fix the problem.

As I mentioned at the start of this white paper, the key to DebugLive is its awesome collaboration capability. Remote debugging is the heart of that collaboration. While remote debugging has been a feature of Visual Studio and WinDBG for years, fundamentally those features are limited by firewalls and other network issues so that they only work well when the machines are on the same domain or behind the same firewall. This is fine for huge multinational companies where the network across continents is completely under their control. Remote debugging with existing debuggers is limited enough that I would venture to say it’s used by less than 5% teams consistently.

Remote debugging should work anywhere. That includes not only from developer machines to test labs, but also if those test labs are on another continent. Remote debugging should also work even if the machine your application is running on is at another company. DebugLive set out to make remote debugging seamless across any connection. If the machines can connect to the internet, you can remote debug across them. There's no need to configure firewalls or open ports anywhere as the connections are over standard HTTPS port 80. This is how remote debugging should work.

The ultimate in remote debugging is to get a customer to set up remote debugging so you can debug the problem they are encountering. All you need to do is create an account for the customer as part of your DebugLive subscription and give them the www.debuglive.com URL. Once they log in you'll have them click the Remoting tab, Start Session button and give the session a name. On your machine, you'll start a DebugLive sessions (Figure 1) and click the Remoting tab, Join Session button. Now you're connected and you can start debugging away. It literally is that simple to debug across the internet.

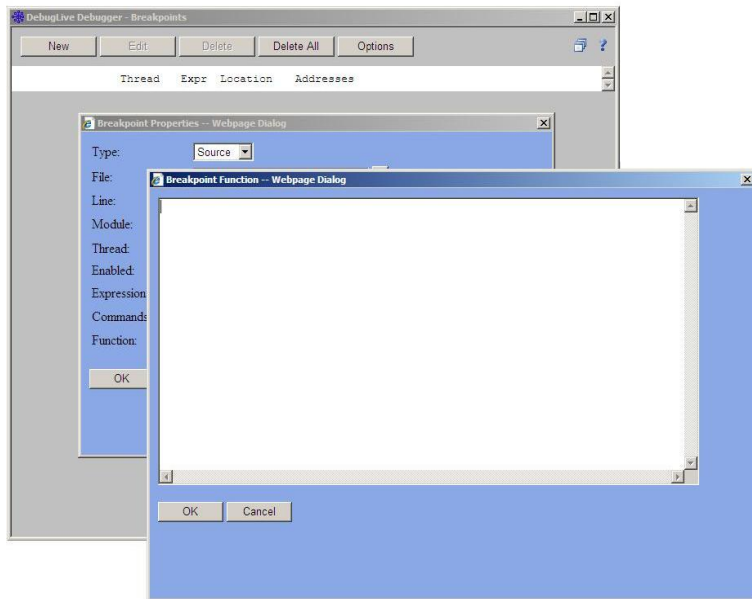
Figure 1. Example DebugLive debugging session.



DebugLive contains all the debugging goodness you would expect from an advanced debugger. Two features in particular stand out for developers. The first is the amazing capability to use JavaScript as the command language for expressions and functions to execute at breakpoints. The key to debugging faster is to program your debugger to stop only when your particular hypothesis occurs. Unfortunately, nearly all developers sit there pressing the Go command hundreds of thousands of times because they haven't taken the time to learn their debugging tools. If you're paid by the hour, that's great for the developer because they can debug their way to a lot of money, but it's a huge waste for the company.

In the following example, I enter a JavaScript function for a breakpoint (Figure 2).

Figure 2. The Breakpoint Function dialog box.



I've applied code at a breakpoint that checks the state of a variable in scope. If the value is zero, somehow my pointer has become corrupted so I need to dump out all the stacks and threads.

```
if ($var("m_pChar") == 0)
{
    // The corruption occurred that set m_pChar to NULL.
    // Take a look at all the call stacks.
    $cmd("stacks -t *");
    /// Dump all the threads.
    $cmd("threadlist");
    // We have to break.
    return true;
}
else
{
    // There was no problem continue without stopping.
    return false;
}
```

This is a simple example. But the idea is that you can do deep programmability when debugging, which opens up all sorts of interesting avenues of analysis.

One of my favorite debugging features in DebugLive is the Command Schedule. Many times when you're debugging you need to see changes in your process over time. A great example of is when you're trying to track down a memory leak. Most memory leaks are those small and sneaky leaks that take thousands of executions to start seeing the effect. When debugging that type of leak, many developers turn to

tools like the Windows Performance Monitor to observe their heap usage. However, Performance Monitor only looks at the application from the outside. What would be far better is that you could see the same information from inside the application where we can look at the appropriate variables and locals.

DebugLive supports a wonderful command called `ScheduleAdd`, which allows you to specify commands that you would like to run on a periodic basis. A perfect example of where the `ScheduleAdd` command makes debugging simple is when dealing with multithreaded applications. Say you are working on an application that isn't scaling as expected. That means you have threads blocking on synchronization objects more than you expect. Because of the nature of multithreading it's very hard to visualize which threads are grabbing what synchronization objects when, especially if you are talking about a multithreaded server application with many threads.

With the `ScheduleAdd` command, you could issue the following command to execute the `stack` command to walk all the call stacks every 30 seconds starting as soon as you execute the command.

```
ScheduleAdd -c "stack -t *" -t now -r 00:00:30
```

Once you have the call stacks you can analyze the parameters looking for those handles or critical sections that are passed to `WaitForSingleObject` or `EnterCriticalSection` respectively. From there you'll find the synchronization object with the most contention. With the excellent ability to schedule commands in DebugLive, it makes all types of analysis possible that would take huge amounts of time to do manually.

While it's great to get those call stacks as I described with the `ScheduleAdd` command, you can execute any DebugLive command when the scheduled time triggers. There are two commands in particular that I want to mention as they open up some very interesting possibilities for diagnosing problems. The first is `ScreenShot`, which as the name implies captures an image of the screen. What's very nice is that you can automatically upload the screen shot to the DebugLive server as part of a problem report. While a picture can say 1,000 words, wouldn't a video be a whole novel?

If you want to see a video of what's happening on the target computer, DebugLive comes with an extension, called, appropriately enough, Video extension, that will capture video. To load the extension click the Load button in the Recording, Video toolbar. Once the extension is loaded, you can record both video and sound of what's happening on the computer. Note that any files you can easily upload any files you create with DebugLive, like screen shots, videos, or mini dumps to the DebugLive servers for anyone else on your team to view and process.

In this white paper I wanted to introduce the idea of DebugLive and discuss how easy it is to set up for just about any environment in the world. When you make it easy to share problem reports across the team and company, the easier it is for problems to get solved. DebugLive brings many interesting ideas to debugging that we've needed for years. With a tool like DebugLive you'll be debugging faster, which means you'll be developing faster, which results in shipping faster. You owe it to your team to check out DebugLive and see what it can do for you.