Needle in a Stack Trace
Dan Foreman, White Star Software

Abstract: Needle in a Stack Trace

Stack Trace files contain valuable information that can be very useful to a Progress developer or Database Administrator. This presentation demonstrates how to understand and analyze Stack Trace files.

Topics include:
• How to safely generate a Stack Trace on all operating systems
• A method for generating Stack Trace files without human intervention
• Analyzing the contents of a Stack Trace file
• Live case studies of how Stack Trace files helped to understand and fix problems
• Alternatives to Stack Trace files
Needle in a Stack Trace

Understanding (maybe) & Finding Value (possibly) in Stack Traces

Dan Foreman, White Star Software (sort of)
danf@prodb.com
Dan Foreman – Short C.V.

• Progress user since 1984

• Author
  – Progress Performance Tuning Guide
  – Progress System Tables
  – Progress Database Administration Guide
  – And lots of other minor publications
  – Electronic media and “traditional” paper are available
Non-Progress Interests

• Travel (75 Countries – mostly on Progress-related trips)
• Basketball
• Cycling
• Photography
  – My best pictures: https://www.flickr.com/gp/154111591@N05/4ngeC3
In computing, a stack trace is a report of the active stack frames at a certain point in time during the execution of a program. Programmers commonly use stack tracing during interactive and post-mortem debugging. End-users may see a stack trace displayed as part of an error message, which the user can then report to a programmer. A stack trace allows tracking the sequence of nested functions called up to the point where the stack trace is generated. In a post-mortem scenario this extends up to the function where the failure occurred (but was not necessarily caused).
In computing, a core dump (in Unix parlance), memory dump, or system dump consists of the recorded state of the working memory of a computer program at a specific time, generally when the program has crashed or otherwise terminated abnormally. In practice, other key pieces of program state are usually dumped at the same time, including the processor registers, which may include the program counter and stack pointer, memory management information, and other processor and operating system flags and information. Core dumps are often used to assist in diagnosing and debugging errors in computer programs.

The name comes from magnetic core memory, the principal form of random access memory from the 1950s to the 1970s. The name has remained long after magnetic core technology became obsolete.
Session Scope

• There are multiple types of “tracing” in Progress such as:
  – “Classic” Stack Traces
  – 4GL Tracing
  – SQL Statement Tracing
  – ODBC Tracing

• This session only covers Classic Stack Traces
History - Part 0

Progress code has had a built in mechanism to generate a core files for a very long time but that feature has evolved dramatically over the years.
In old versions of Progress the only way to get the core file was by sending a signal to the process to terminate and dump core.

The signal was sent as a special bit that is turned on in a message, i.e. a PROMSGS Message.

Progress could create a message that would go to the screen or database log or both and, if this bit was set, it would cause the process to exit and the core file to be generated.
Some Control Codes from PROMSGS

- `%B` Write message to both the log file and the screen.
- `%c` Long table name. (Example #3113)
- `%C` Table name.
- `%d` Substitute a Decimal value.
- `%D` Dbkey value. (Example #1124)
- `%E` Include the error# from the operating system. See /usr/include/sys/errno.h for a list of errors
- `%F` Field name.
- `%g` Stands for 'gronk'. Crash the process.
- `%G` *Stands for Big Gronk. Crash the process and generate a core file.*
- `%i` Substitute an Integer value.
- `%k` Key labels. (Example #112, #600)
- `%K` Keyboard response required.
Some Control Codes from PROMSGS

- `%l` Display a long integer. (Example #761)
- `%L` Write to the log file only.
- `%M` Don’t put new line in the error message.
- `%n` Number. (Example #1676)
- `%N` Field name.
- `%p` Pointer. (Example #1448, #1545)
- `%r` Put a new line into the message output. (Example #2728, #2776)
- `%s` Substitute a String.
Some Control Codes from PROMSGS

- %t  Print one byte. (Example #760, #1280)
- %T  Tab. (Example #3084, #3089 which don't exist in V10/V11).
- W   Warning level. (Example #1880, #2750)
- %x  Numeric value. (Example #9382)
- %X  Hex value. (Example #3644, #3648)
History - Part 2

- To get something useful out of a core file (which is just a bunch of numbers), the following are required:
  - The exact copy of the executable that created the core file
  - A debugger
- So for customers with hot fixes, HLC (anyone ever use that feature?), etc. it became impossible for Progress to read the core files
- So enter the prostack command
prostack

• Not to be confused with ProGetStack which is covered later
• prostack is a shell script located in $DLC/bin
• prostack doesn’t exist on Windows
• Syntax
  – prostack -r ImageFile CoreFile
  – prostack -a Pid ImageFile
• A little complex because a debugger such as adb or gdb is required
prostack Operation

1. The user is prompted to select an option which best describes the OS and machine that the user is logged into.

2. The user is prompted to confirm the selected option. If the user does not confirm the selected option, "prostack" will prompt the user again with the original OS / machine selection list.

3. Log messages are displayed on the screen which indicate what actions "prostack" is working on.

4. The stack trace output is appended to the <debugger-name>.log file in the user's working directory. The log file is created if it does not already exist.
HP (in the era when HP was a very popular Progress platform) provided Progress with a call to generate a stack trace at runtime, i.e. without terminating the process. Once HP provided this feature, Mike implemented the traceback to create a protrace.pid# file. This was a very useful feature, so Progress asked other hardware vendors if they could do the same. Over time, maybe a year or two, all of them came up with something.
This feature allowed Progress to get a stack trace when a core file was created. However, it became clear that Progress also needed to figure out what a hung process was doing, without having it exit, because if it was connected to shared memory, then at the time it could have crashed the database because it was holding a latch.
So the next thing Mike did was modify the Progress signal handling code to allow a specific signal to generate the protrace file without terminating the process.

Then Tim Sargent (long time Progress 4GL developer) approached Mike and said “this is excellent, why don’t we dump the 4GL stack as well”? He showed Mike how to do it and it was then incorporated into the traceback code.
History - Part 5 – courtesy Mike Furgal

So while all this is good – this functionality has been abused. All of the code that came from the vendors is not reentrant, meaning if you are in the code, you cannot get a signal to come into the code again. For example you are dumping a stack trace and you get another signal to dump the stack trace. In that case, the process will likely terminate. Some Progress customers have been doing this to the extent that they run proGetStack every minute to trace what 4GL code the process is executing, and this has had some bad side-effects. Recently (V11 era) the code has changed to be more robust and to be reentrant.
Stack Traces are just one of many tools

• Without Progress source code a Stack Trace might be of zero value
• On the other hand they can be of excellent value
• Use them for the right problem; in general works well for:
  – Sessions/Processes Hanging
  – Non-responsive processes
  – Run away processes
  – Limbo DB connections (disconnected from DB but not really)
Stack Trace Basics

• The Stack Trace displayed in descending order in time, i.e. from most recent (top) to least recent (bottom); this is also known as a traceback

• The output appears to imply hierarchical & serial processing but the reality can be much more complex

• A typical stack trace can contain a mixture of 4GL/ABL related activity as well as Database information:
  – “C Stack Trace”
  – “ABL Stack Trace”
Essential Info - *ix

• kill –SIGUSR1

• The numeric value of SIGUSR1 varies by platform ("kill -l", with a lower case "ell", will list the local names and values)
Basics – Manual Generation

- proGetStack – because there is no analogous *kill* command on Windows
- proGetStack also exists on *ix for platform consistency
- Need to be root (or Admin on Windows) or same group as the process that needs to be probed
Essential Info

- It is possible to send the stack trace signal to almost any Progress command.
- proGetStack should not terminate a process; at least it is not intended to do so. The SIGUSR1 signal handler does not try to terminate the AVM. But there have been situations where generating the protrace information “stumbles” over a memory stomp or some other unexpected circumstance, which would then result in a segfault, thus terminating the process.
Essential Info

- A protrace file will be overwritten, not appended
- The output will be different between REMC (Remote Client) and SELF (Self-Service Client) database activities. There shouldn’t be any Latch activity for REMC processes because they don’t attach to Shared Memory
Basics - Windows

- Windows Stack Traces are not as comprehensive as *ix

```bash
proenv>progetstack 9604 (promon,_mprosrv)
OpenEdge Release 11.5 as of Fri Dec 5 19:02:15 EST 2014
Cannot get stack of process 9604.
The process id you specified may be invalid, not an OpenEdge ABL process or is a process from a pre 10.1C release of OpenEdge.
```
Essential Info

- Anything in the fd (Federated Database) layer & below is database related, not ABL/4GL
- If there is no fd* tokens then there is no DB involvement
- Exceptions – single user skips some stuff
- Temp-Tables are an exception too... just another (unique) DB type
- The names & formatting are OS vendor specific...
- Some utilities will ignore hangup signals... i.e. ignore stack trace requests, i.e. kill –SIGUSR1 is not guaranteed to be consistent
Multiple Trace/Core Files

- Sometimes multiple Stack Trace files will be dumped. Example a DB with a corrupt RM (Record Manager) Block Header (#1124 error) produced:
  - protrace.<brokerPID> from the Broker
  - procore.<clientPID> from the SELF Client
  - protrace.<clientPID> from the (same) SELF Client
  - A hex dump of the corrupt block in the Database Log (.lg) file
- The PID appended to the file name correlates to the PID written to the various entries in the DB log file
PROGRESS stack trace as of Mon Oct 30 16:41:04 2017

Progress OpenEdge Release 11.6 build 1346 SP02 on Linux prodb 2.6.32-504.el6.x86_64 #1 SMP Wed Oct 15 04:27:16 UTC 2014

Command line arguments are

Startup parameters:
Stack Trace – Main Sections – Part 2

<C Stack Trace> (doesn’t have a label)

** ABL Stack Trace **

** Persistent procedures/Classes **

** PROPATH **

** Databases (logical/type/physical) **

White Star Software
Stack Trace – Manual Generation

When a Stack Trace is manually initiated (with kill or ProGetStack) the following lines are usually the first part (but last executed) of the Stack Trace

#1 [0xad3d2b] uttraceback+0xfb from /opt/dlc/dlc11.6.2/bin/_progres
#2 [0xad71ce] uttrace+0x13e from /opt/dlc/dlc11.6.2/bin/_progres
#3 [0x4dd129] drProTrace+0x3c from /opt/dlc/dlc11.6.2/bin/_progres
#4 [0x4dc7d9] drSigDo1+0x8e from /opt/dlc/dlc11.6.2/bin/_progres
#5 [0x4dc9a6] drSigDispatch+0xf2 from /opt/dlc/dlc11.6.2/bin/_progres
<table>
<thead>
<tr>
<th>Prefix</th>
<th>Long Name</th>
<th>Example</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>bf</td>
<td>Client Record Buffer</td>
<td>bfFindRow</td>
<td>Fetch a database record. Client side.</td>
</tr>
<tr>
<td>bk</td>
<td>Database Block</td>
<td>bkGetStamp</td>
<td>Bkrepl replacing bytes in a block.</td>
</tr>
<tr>
<td>bm</td>
<td>Buffer Manager (-B)</td>
<td>bmsteal</td>
<td>Read a DB block into a buffer.</td>
</tr>
<tr>
<td>db</td>
<td>Database</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dr</td>
<td>??</td>
<td>drexit</td>
<td>Nobody @ Progress remembers what ‘dr’ means. Generally used for both setup/startup and cleanup/exit. Abused &amp; misused over time.</td>
</tr>
<tr>
<td>dsm</td>
<td>DB Storage Manger</td>
<td></td>
<td>Entry points for DB Storage Manger APIs</td>
</tr>
<tr>
<td>fd</td>
<td>Federated Databases</td>
<td>fdfind</td>
<td>Correlates to a 4GL FIND statement; Oracle, ODBC, etc.</td>
</tr>
<tr>
<td>fm</td>
<td>Function Manager</td>
<td>fmfldu</td>
<td>4GL Layer</td>
</tr>
<tr>
<td>Prefix</td>
<td>Long Name</td>
<td>Example</td>
<td>Remarks</td>
</tr>
<tr>
<td>--------</td>
<td>------------------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>io</td>
<td>Input-Output</td>
<td>iopause</td>
<td></td>
</tr>
<tr>
<td>lat</td>
<td>Latch Activity</td>
<td>latSleep</td>
<td>Normally associated with SELF-Service Clients</td>
</tr>
<tr>
<td>lk</td>
<td>Record Lock</td>
<td>lkBusy</td>
<td>DB Code… complex interaction with 4GL code in SELF mode</td>
</tr>
<tr>
<td>msg</td>
<td>Message Handler</td>
<td>msgout</td>
<td>For writing messages to the screen, the log, or both</td>
</tr>
<tr>
<td>nc</td>
<td>Network Client</td>
<td>ncdsmObjectLock</td>
<td>I have only seen these for Remote Client Stack Traces....</td>
</tr>
<tr>
<td>ns</td>
<td>Network Server</td>
<td></td>
<td>_mprosrv</td>
</tr>
<tr>
<td>om</td>
<td>Object Manager</td>
<td>omGetObjectRecord</td>
<td></td>
</tr>
<tr>
<td>pro</td>
<td>Progress DB</td>
<td>proTxsavepoint</td>
<td>Below the fd layer, i.e. a Progress DB</td>
</tr>
<tr>
<td>rl</td>
<td>Recovery Log</td>
<td>rIWriteTxBegin</td>
<td>Write a Transaction Start note to the BI file; After Image stuff also</td>
</tr>
</tbody>
</table>
## Stack Trace Name Translation

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Long Name</th>
<th>Example</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>rm</td>
<td>Record Manager</td>
<td>rmLocate</td>
<td>Rocket Engine</td>
</tr>
<tr>
<td>rn</td>
<td>4GL Runtime</td>
<td>rnexec</td>
<td>4GL/ABL code</td>
</tr>
<tr>
<td>sl</td>
<td>Schema Layer</td>
<td>slrmget</td>
<td>Nobody @ Progress seems to remember what ‘sl’ stands for</td>
</tr>
<tr>
<td>tm</td>
<td>Transaction Manager</td>
<td>tmMarkSavePoint</td>
<td>Rocket Engine</td>
</tr>
<tr>
<td>um</td>
<td>User interface Manager</td>
<td>umeDispatchEvent</td>
<td>4GL related</td>
</tr>
<tr>
<td>ut</td>
<td>Utility</td>
<td></td>
<td>General purpose code</td>
</tr>
<tr>
<td>wv</td>
<td>Character mode UI</td>
<td>wvAlertBox</td>
<td>Related to user input; FRAME LAYOUT</td>
</tr>
</tbody>
</table>
06/26/02 07:46:05 [1660]

Progress Recent Message(s):
(1124) (10561) (10560) (4229) (9445)

SYSTEM ERROR: wrong dbkey in block. Found 0, should be 24320 (1124)
Writing block 189 to log file. Please save and send the log file to Progress Software Corp. for investigation. (10561)

bkRead: Error occurred in area 10, block number: 189, extent: C:\class\dbag\proc\msg\msg91d_10.d1. (10560)
Corrupt block detected when reading from database. (4229)
SYSTEM ERROR: read wrong dbkey at offset 1556480 in file C:\class\dbag\proc\msg\msg91d_10.d1 found 0, expected 24320, retrying. (9445)

*** 4GL Call Stack ***

Last action: BLOCK HEADER  (2)
65535: ../msgquery.p (..\msgquery.p)
1: C:\class\dbag\proc\msg\p16941r.ped (C:\class\dbag\proc\msg\p16941r.ped)
652: adecomm/_runcode.p (adecomm/_runcode.r)
3818: ExecuteRun adeedit/_proedit.p (adeedit/_proedit.r)
10636: RunFile adeedit/_proedit.p (adeedit/_proedit.r)
2323: USER-INTERFACE-TRIGGER adeedit/_proedit.p (adeedit/_proedit.r)
11871: adeedit/_proedit.p (adeedit/_proedit.r)
453: _edit.p (c:\DLC91D\tty\_edit.r)
V10 Example

(0) 0x4000000000166fa0  uttraceback + 0x60 [/usr/dlc/bin/_mprshut]
(1) 0x400000000014a250  utcore + 0x3a0 at /vobs_prgs/src/ut/utbuf.c:603 [/usr/dlc/bin/_mprshut]
(2) 0x4000000000149560  dreetix + 0x540 at /vobs_prgs/src/drsys/drsetup.c:929 [/usr/dlc/bin/_mprshut]
(3) 0x400000000003c0770  drSigFatal + 0xf0 at /vobs_rkt/src/glue/drsig.c:1199 [/usr/dlc/bin/_mprshut]
(4) 0xe0000001201cb440  ---- Signal 8 (SIGFPE) delivered ----
(5) 0xc000000000550190  _poll_sys + 0x30 [/usr/lib/hpux64/libc.so.1]
(6) 0xc0000000005647a0  _poll + 0x80 at ../../../../../core/libs/libc/shared_em_64_perf../core/syscalls/t_poll.c:26
(7) 0x4000000000359be0  latSleep + 0x190 at /vobs_rkt/src/dbmgr/lat/latctl.c:1290 [/usr/dlc/bin/_mprshut]
(8) 0x400000000035b7d0  latObjLatchLock + 0x320 at /vobs_rkt/src/dbmgr/lat/latctl.c:873 [/usr/dlc/bin/_mprshut]
(9) 0x40000000001afe50  bmLocateBuffer2 + 0x600 at /vobs_rkt/src/dbmgr/bm/bmbuf.c:2868 [/usr/dlc/bin/_mprshut]
(10) 0x400000000018f180  bkGetStamp + 0x150 at /vobs_rkt/src/dbmgr/bk/bkrec.c:306 [/usr/dlc/bin/_mprshut]
(11) 0x400000000026d160  dsmUserConnect + 0x360 at /vobs_rkt/src/dbmgr/dsm/dsmuser.c:151 [/usr/dlc/bin/_mprshut]
(12) 0x4000000000407f70  dsmmain_doit + 0x430 at /vobs_rkt/src/dbadmin/drshut.c:1702 [/usr/dlc/bin/_mprshut]
(13) 0x40000000000f1600  main + 0x150 at /vobs_rkt/src/dbadmin/drshut.c:1202 [/usr/dlc/bin/_mprshut]
(14) 0xc00000000006ea30  main_opd_entry + 0x50 [/usr/lib/hpux64/dld.so]
V11, Editor session, No DB connections

PROGRESS stack trace as of Thu Oct 26 15:31:38 2017

Progress OpenEdge Release 11.5 build 1114 on WINNT

Startup parameters:
- pf c:\DLC\11.5.0\startup.pf, -cpinternal ISO8859-1, -cpstream ISO8859-1, -cpcoll Basic, -cpcase Basic, -d mdy, -numsep 44, -numdec 46, (end .pf), -1

** ABL Stack Trace **

--> adeedit/_proedit.p at line 12280 (adeedit/_proedit.r)
    _edit.p at line 408 (c:\DLC\11.5.0\tty\_edit.r)

** Persistent procedures/Classes **

** PROPATH **
., c:\DLC\11.5.0\tty\ablunit.pl, c:\DLC\11.5.0\tty\adecomm.pl, c:\DLC\11.5.0\tty\adecompi.pl, c:\DLC\11.5.0\tty\adecomp.pl, c:\DLC\11.5.0\tty\adeedit.pl, c:\DLC\11.5.0\tty\adeshar.pl, c:\DLC\11.5.0\tty\dataadmin.pl, c:\DLC\11.5.0\tty\OpenEdge.BusinessLogic.pl, c:\DLC\11.5.0\tty\OpenEdge.Core.pl, c:\DLC\11.5.0\tty\OpenEdge.ServerAdmin.pl, c:\DLC\11.5.0\tty\prodict.pl, c:\DLC\11.5.0,c:\DLC\11.5.0\bin

** Databases (logical/type/physical) **

End of Protrace
Waiting for a Quiet Point Latch - Windows

==================================================================================================================================
PROGRESS stack trace as of Thu Oct 26 15:39:57 2017
==================================================================================================================================

Progress OpenEdge Release 11.5 build 1114 on WINNT

** ABL Stack Trace **

--> chg.p at line 1 (.\chg.p)

** Persistent procedures/Classes **

** Databases (logical/type/physical) **
  msg       PROGRESS  msg

** End of Protrace **
Waiting for a Quiet Point Latch - *ix

<snip>

#6 [0x9ba0f710] _fini+0xfffe5f0 from /lib64/libpthread.so.0
#7 [0x9bb6df178] __rpc_thread_destroy+0xfff9dce8 from /lib64/libc.so.6
#8 [0x935137] utnap+0x27 from /opt/dlc/dlc11.6.2/bin/progres
#9 [0x9ddbc9] latSleep+0x9df from /opt/dlc/dlc11.6.2/bin/progres
#10 [0x9de4db] latXlock+0xe5 from /opt/dlc/dlc11.6.2/bin/progres
#11 [0x9de7d2] latlatch+0xe4 from /opt/dlc/dlc11.6.2/bin/progres
#12 [0x9dd95f9] rlWriteTxBegin+0x16d from /opt/dlc/dlc11.6.2/bin/progres
#13 [0x9970705] rLogAndDo+0x5a from /opt/dlc/dlc11.6.2/bin/progres
#14 [0x9bf93f] tmMarkSavePoint+0x124 from /opt/dlc/dlc11.6.2/bin/progres
#15 [0x94c691] dsmTransaction+0x1a6 from /opt/dlc/dlc11.6.2/bin/progres
#16 [0x72bc8] proTxSavepoint+0x58 from /opt/dlc/dlc11.6.2/bin/progres
#17 [0x46dcaef] fdTxSavepoint+0x27e from /opt/dlc/dlc11.6.2/bin/progres
#18 [0x46e409] fdkeychg+0x59 from /opt/dlc/dlc11.6.2/bin/progres
#19 [0x7324e9] proffdu+0x7c9 from /opt/dlc/dlc11.6.2/bin/progres
#20 [0x46b161] fdffldu+0x131 from /opt/dlc/dlc11.6.2/bin/progres
#21 [0x7ad057] fmfdiu+0x107 from /opt/dlc/dlc11.6.2/bin/progres
#22 [0x7bed03] rmqopns+0x1493 from /opt/dlc/dlc11.6.2/bin/progres
#23 [0x768632] rnupd_2+0xd2 from /opt/dlc/dlc11.6.2/bin/progres
#24 [0x7f929d] rnterpret+0x27 from /opt/dlc/dlc11.6.2/bin/progres
#25 [0x633eb8] rnmDispatchEvent+0x86b from /opt/dlc/dlc11.6.2/bin/progres
#26 [0x7f92d7] umeDispatchEvent+0x86b from /opt/dlc/dlc11.6.2/bin/progres
#27 [0x75513b] umeDispatchEvent+0x86b from /opt/dlc/dlc11.6.2/bin/progres
#28 [0x76605f] rnwaitfor+0x32f from /opt/dlc/dlc11.6.2/bin/progres
#29 [0x777171] rnexec_entry+0x191 from /opt/dlc/dlc11.6.2/bin/progres
#30 [0x7f92d7] rnterpret+0x27 from /opt/dlc/dlc11.6.2/bin/progres
#31 [0x4bf65b] rnrq+0x80 from /opt/dlc/dlc11.6.2/bin/progres
#32 [0x4501f2] main+0x472 from /opt/dlc/dlc11.6.2/bin/progres
#33 [0x9966ed5d] __rpc_thread_destroy+0xfffed8ad from /lib64/libc.so.6
**Broker Startup – Corrupt DB Block**

<table>
<thead>
<tr>
<th>#</th>
<th>Address</th>
<th>Function Name</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>84</td>
<td>0x424911</td>
<td>drexit+0x1d1</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>85</td>
<td>0x42128d</td>
<td>msgout+0x20d</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>86</td>
<td>0x422450</td>
<td>msgnCB+0xa0</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>87</td>
<td>0x422548</td>
<td>msgCB+0x88</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>88</td>
<td>0x50ba81</td>
<td>dsmFatalMsgrCallBack+0xddd</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>89</td>
<td>0x499f58</td>
<td>bkRead+0x3e9</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>90</td>
<td>0x4a2099</td>
<td>bmrdnew+0x1b8</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>91</td>
<td>0x4a3fa2</td>
<td>bmsteal+0x53e</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>92</td>
<td>0x4a4c2c</td>
<td>bmLocateBuffer2+0xa30</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>93</td>
<td>0x577d3d</td>
<td>rmLocate+0x131</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>94</td>
<td>0x578283</td>
<td>rmRecordFetch+0x183</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>95</td>
<td>0x53d2c5</td>
<td>omGetObjectRecord+0x7c</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>96</td>
<td>0x53eae9</td>
<td>omCachePreload+0x48b</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>97</td>
<td>0x4e58ca</td>
<td>dbSetOpen+0x5df</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>98</td>
<td>0x4e85e9</td>
<td>dbenv1+0x1264</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>99</td>
<td>0x4e8a44</td>
<td>dbenv+0x56</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>100</td>
<td>0x51e2ac</td>
<td>dsmUserSecureConnect+0x7bd</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>101</td>
<td>0x47805d</td>
<td>doserve+0x7a2</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
<tr>
<td>102</td>
<td>0x478751</td>
<td>main+0x103</td>
<td>/opt/dlc/dlc11.6.2/bin/_mprosrv</td>
</tr>
</tbody>
</table>
Single User – Corrupt DB Block

#4  [0x461012] drexit+0x212 from /opt/dlc/dlc11.6.2/bin/_progres
#5  [0x451f6d] msgout+0x20d from /opt/dlc/dlc11.6.2/bin/_progres
#6  [0x453130] msgCB+0xa0 from /opt/dlc/dlc11.6.2/bin/_progres
#7  [0x453228] msgCB+0x88 from /opt/dlc/dlc11.6.2/bin/_progres

#8  [0x93c4b1] dsmFatalMsgnCallBack+0xdd from /opt/dlc/dlc11.6.2/bin/_progres
#9  [0x8ca988] bkRead+0x3e9 from /opt/dlc/dlc11.6.2/bin/_progres
#10 [0x8d2ac9] bmrdnew+0x1b8 from /opt/dlc/dlc11.6.2/bin/_progres
#11 [0x8d49d2] bmsteal+0x53e from /opt/dlc/dlc11.6.2/bin/_progres
#12 [0x8d565c] bmLocateBuffer2+0xa30 from /opt/dlc/dlc11.6.2/bin/_progres
#13 [0x9a876d] rmLocate+0x131 from /opt/dlc/dlc11.6.2/bin/_progres
#14 [0x9a8cb3] rmRecordFetch+0x183 from /opt/dlc/dlc11.6.2/bin/_progres
#15 [0x91b17d] dbRecordGet+0x37c from /opt/dlc/dlc11.6.2/bin/_progres
#16 [0x948529] dsmRecordGet+0xde from /opt/dlc/dlc11.6.2/bin/_progres
#17 [0x4d7e0a] slrmget+0x58f from /opt/dlc/dlc11.6.2/bin/_progres
#18 [0x4da1b2] dbrmget+0x15d from /opt/dlc/dlc11.6.2/bin/_progres
#19 [0x4db77c] dbnxt1+0x58a from /opt/dlc/dlc11.6.2/bin/_progres
#20 [0x44d9b0] dbnxt+0xbf from /opt/dlc/dlc11.6.2/bin/_progres
#21 [0x73725f] profnd+0x58f from /opt/dlc/dlc11.6.2/bin/_progres
#22 [0x73a145] pro1ksch+0x195 from /opt/dlc/dlc11.6.2/bin/_progres
#23 [0x72f741] prolksch+0x195 from /opt/dlc/dlc11.6.2/bin/_progres
#24 [0x72f74c] proPostConnectionSetup+0x446 from /opt/dlc/dlc11.6.2/bin/_progres
#25 [0x73106d] procon+0x4fd from /opt/dlc/dlc11.6.2/bin/_progres
#26 [0x73919d] sccon+0x20a from /opt/dlc/dlc11.6.2/bin/_progres
#27 [0x729e36] scconx+0x20a from /opt/dlc/dlc11.6.2/bin/_progres
#28 [0x729e3b] scconx+0x20a from /opt/dlc/dlc11.6.2/bin/_progres
#29 [0x729e3c] sccon+0x20a from /opt/dlc/dlc11.6.2/bin/_progres
#30 [0x729e40] scon+0x20a from /opt/dlc/dlc11.6.2/bin/_progres
#31 [0x729e49] scon+0x20a from /opt/dlc/dlc11.6.2/bin/_progres
#32 [0x729e50] scon+0x20a from /opt/dlc/dlc11.6.2/bin/_progres
#33 [0x729e51] scon+0x20a from /opt/dlc/dlc11.6.2/bin/_progres
#34 [0x729e52] scon+0x20a from /opt/dlc/dlc11.6.2/bin/_progres

White Star Software
According to Roy Ellis, there is no way to get a stack trace using kill with the new PASOE agent.

There is a way to get stack information using the OE Manager REST API for just one session, but OE Manager needs to be running.

You can get a stack trace by connecting to the process with gdb (GNU Debugger) and then ask for a trace; This will give you a stack trace for every thread in the agent. Although you could also look at each thread, but there is no way to relate a thread in gdb to the session in the agent you can see through OE Manager REST API.
SQL Trace Files

- sqlexp does not recognize kill –SIGUSR1 because it’s written Java
- _sqlsrv2 does respond (example on the next slide)
SQL Trace File from _sqlsrv2

#1 [0x8a84c2] dbut_uttraceback+0x12b  from /dlc11.6/bin/_sqlsrv2
#2 [0x8a805b] dbut_uttrace+0x139  from /dlc11.6/bin/_sqlsrv2
#3 [0x838a29] drProTrace+0x3c   from /dlc11.6/bin/_sqlsrv2
#4 [0x837822] drSigDo1+0x1c    from /dlc11.6/bin/_sqlsrv2
#5 [0x837b06] drSigDispatch+0xa4 from /dlc11.6/bin/_sqlsrv2
#6 [0x9ba0f710] _fini+0xfffffffe5f0  from /lib64/libpthread.so.0
#7 [0x9b6df1b3] __rpc_thread_destroy+0xfff9dd03 from /lib64/libc.so.6
#8 [0x7949d9] _Z18os_select_read_1fdijRiS_S_+0x53 from /dlc11.6/bin/_sqlsrv2
#9 [0x42539c] _ZN16listening_port_t21query_pending_connectEP5log_tiRijhPhS3_S3_S3_S3_hiS3_+0x70 from /dlc11.6/bin/_sqlsrv2
#10 [0x42a2db] _ZN12server_ctl_t12serve_eventsEv+0x68f from /dlc11.6/bin/_sqlsrv2
#11 [0x42956e] main+0x12a2  from /dlc11.6/bin/_sqlsrv2
#12 [0x9b61ed5d] __rpc_thread_destroy+0xffeedd8ad  from /lib64/libc.so.6
Resources

• Knowledgebase Articles
  – P112486 - HOW TO PRODUCE A STACK TRACE FOR A RUNNING OPENEDGE PROCESS WITHOUT KILLING IT
Thank You
Contact Info

• [email: danf@prodb.com]
• +1 541 908 3437 (but not right now please)