Embedded Application Performance Instrumentation

(aka Rational Application Performance Investigation Details…)

Tom Bascom, White Star Software

Abstract: You can only make an application go faster by improving the parts that are slow. It doesn't help to optimize code that is already fast enough or that is only rarely used.

User table stats, temp table usage stats and the application profiler can all be easily embedded in your application to provide powerful insight into exactly where performance problems are originating.

In this session we will explore these 3 under-utilized OpenEdge features, explain why they are useful and how you can embed them into your own code base to focus your performance improvement activities on code that really is slow rather than wasting time trying to guess which bits aren't as fast as they could be! You will take home simple but powerful code samples that can be easily used with any OpenEdge environment: .NET GUI, Webspeed, App server, PASOE, Character, Batch or anything else!

Embedded Application Performance Instrumentation

aka: Rational Application Performance Investigation Details ☺

Tom Bascom, White Star Software
tom@wss.com

White Star Software

A Few Words about the Speaker

- Tom Bascom: Progress user & roaming DBA since 1987
- Partner: White Star Software & DBAppraise, LLC
  - Expert consulting services related to all aspects of Progress and OpenEdge.
  - Remote database management service for OpenEdge.
  - Author of: 
  - Simplifying the job of managing and monitoring the world's best business applications.
  - **tom@wss.com**

Agenda

- Why?
- The Profiler
- User Table and Index Statistics
- Temp Table Statistics
- What is This “Embedded” Nonsense?

Why **NOT** just use ETIME() and MESSAGE???

- Non-Repeatable
- Subject to a host of external factors
  - CPU speed, disk throughput, virtualization, contention from other user activity, buffer cache efficiency, phase of the moon
- Granularity is too gross (only millisecond timings)
- You have to guess the right target code to measure
- You have to write a lot of custom logging code
- Coding in your production environment is a **BAD** idea

Why?

Target the largest response time component of the most important Business process first.

The performance enhancement possible with a given improvement is limited by the fraction of the execution time that the improved feature is used.

-- Amdahl's Law

The Ultimate KPI is User Experience no amount of charts and graphs showing how well your system is performing matter if the users are not happy.

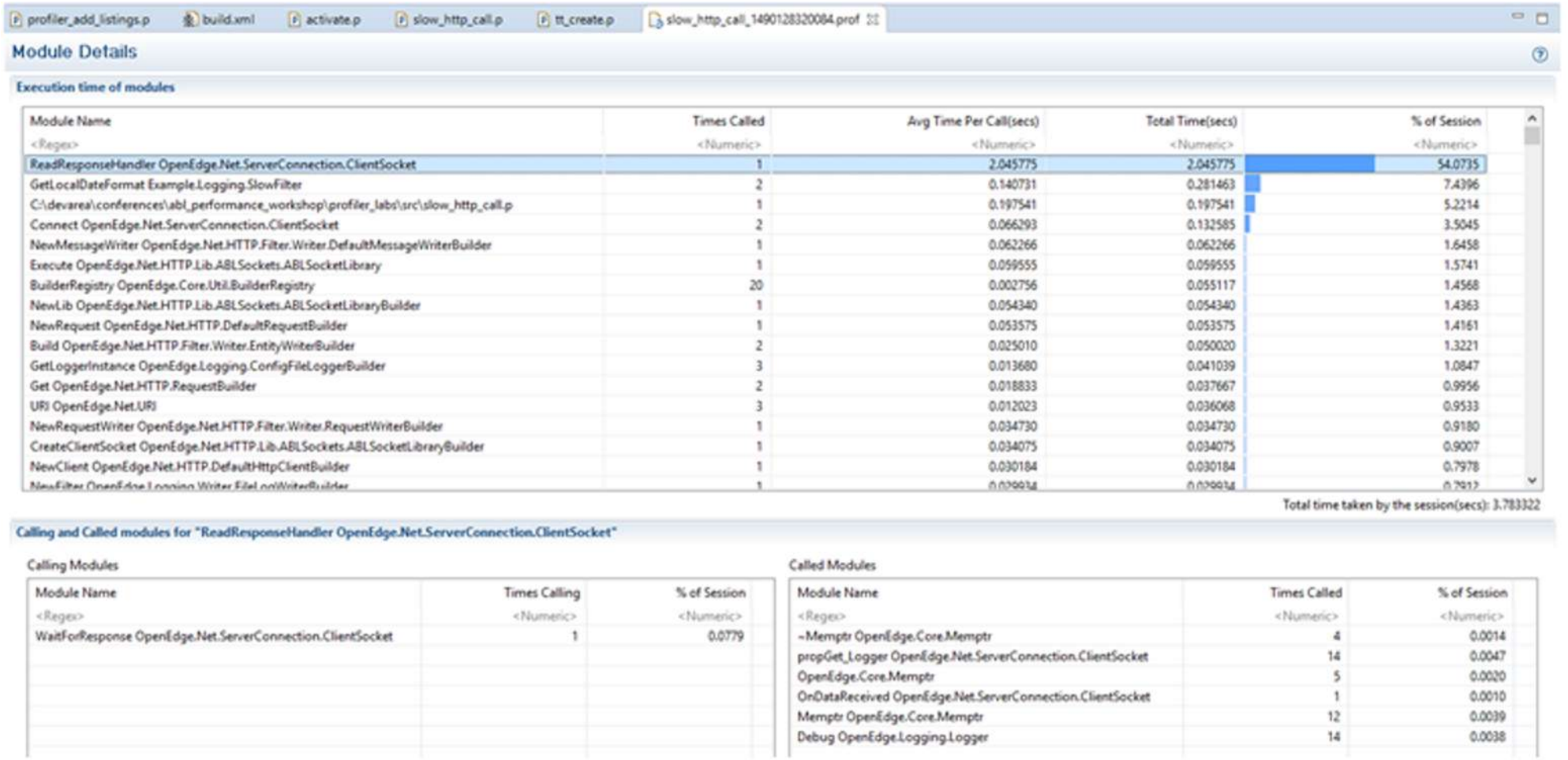
The Profiler

The Profiler

- Microsecond timings of actual code execution time!
  - Does not include waiting for IO (disk IO, sockets, UPDATE, PAUSE etc statements…)
  - Covers ALL of the code!
- First introduced with version 8.2 (-zprofile)
- Greatly Improved with version 9.0 (profiler: handle)
- “Unsupported” is a myth (“lightly documented” is not a myth)
    - The profiling capability *is* supported, the example GUI tool to evaluate the results was not. That tool is also completely unnecessary eye candy.

Unsupported Profiling Eye Candy

New PDSOE Eye Candy



profiler_add_listings.p | build.xml | activate.p | slow_http_call.p | tt_create.p | slow_http_call_1490128320084.prof

Module Details

Execution time of modules

Module Name	Times Called	Avg Time Per Call(secs)	Total Time(secs)	% of Session
<Regex>	<Numeric>	<Numeric>	<Numeric>	<Numeric>
ReadResponseHandler OpenEdge.Net.ServerConnection.ClientSocket	1	2.045775	2.045775	54.0735
GetLocalDateFormat Example.Logging.SlowFilter	2	0.140731	0.281463	7.4396
C:\devarea\conferences\abl_performance_workshop\profiler_labs\src\slow_http_call.p	1	0.197541	0.197541	5.2214
Connect OpenEdge.Net.ServerConnection.ClientSocket	2	0.066293	0.132585	3.5045
NewMessageWriter OpenEdge.Net.HTTP.Filter.Writer.DefaultMessageWriterBuilder	1	0.062266	0.062266	1.6458
Execute OpenEdge.Net.HTTP.Lib.ABLSockets.ABLSocketLibrary	1	0.059555	0.059555	1.5741
BuilderRegistry OpenEdge.Core.Util.BuilderRegistry	20	0.002756	0.055117	1.4568
NewLib OpenEdge.Net.HTTP.Lib.ABLSockets.ABLSocketLibraryBuilder	1	0.054340	0.054340	1.4363
NewRequest OpenEdge.Net.HTTP.DefaultRequestBuilder	1	0.053575	0.053575	1.4161
Build OpenEdge.Net.HTTP.Filter.Writer.EntityWriterBuilder	2	0.025010	0.050020	1.3221
GetLoggerInstance OpenEdge.Logging.ConfigFileLoggerBuilder	3	0.013680	0.041039	1.0847
Get OpenEdge.Net.HTTP.RequestBuilder	2	0.018833	0.037667	0.9956
URI OpenEdge.Net.URI	3	0.012023	0.036068	0.9533
NewRequestWriter OpenEdge.Net.HTTP.Filter.Writer.RequestWriterBuilder	1	0.034730	0.034730	0.9180
CreateClientSocket OpenEdge.Net.HTTP.Lib.ABLSockets.ABLSocketLibraryBuilder	1	0.034075	0.034075	0.9007
NewClient OpenEdge.Net.HTTP.DefaultHttpClientBuilder	1	0.030184	0.030184	0.7978
NewFilter OpenEdge.Logging.Writer.FileLogWriterBuilder	1	0.029934	0.029934	0.7912

Total time taken by the session(secs): 3.783322

Calling and Called modules for "ReadResponseHandler OpenEdge.Net.ServerConnection.ClientSocket"

Calling Modules

Module Name	Times Calling	% of Session
<Regex>	<Numeric>	<Numeric>
WaitForResponse OpenEdge.Net.ServerConnection.ClientSocket	1	0.0779

Called Modules

Module Name	Times Called	% of Session
<Regex>	<Numeric>	<Numeric>
~Memptr OpenEdge.Core.Memptr	4	0.0014
propGet_Logger OpenEdge.Net.ServerConnection.ClientSocket	14	0.0047
OpenEdge.Core.Memptr	5	0.0020
OnDataReceived OpenEdge.Net.ServerConnection.ClientSocket	1	0.0010
Memptr OpenEdge.Core.Memptr	12	0.0039
Debug OpenEdge.Logging.Logger	14	0.0038

Using the Profiler

- -profile startup parameter
  - Non-intrusive
  - Non-selective (all or nothing)
  - Must end session cleanly – no aborts or kills!
- profiler: handle
  - Selective
  - Can turn it on or off at will
  - Can flush the data when **you** want to
  - But requires code insertion or tooling of some sort

Minimal Embedded Usage

```
assign                                  /* enable the profiler        */
  profiler:enabled = yes
  profiler:profiling = yes
.

do i = 1 to 1000000:                    /* some code to be profiled!  */
end.

assign                                  /* disable the profiler       */
  profiler:enabled = no
  profiler:profiling = no
.

profiler:write-data().                  /* create the output file     */
```

PROFILER Attributes

- DESCRIPTION – optional text describing this session
- LISTINGS – whether or not to create debug listings
- DIRECTORY – where to create debug listings (default to –T)
- FILE-NAME – name of output file (default profile.out)
- ENABLED – yes/no; initializes listings and so forth
- PROFILING – turn profiling on or off

Other PROFILER Attributes

- TRACE-FILTER – CSV list of “**matches**” criteria for procedure tracing
- TRACING – line level tracing
- COVERAGE - % coverage support

PROFILER Methods

- Write-Data() – flush accumulated data to output file.
- User-Data(char) – write user defined data, such as VST statistics, to the output file.

Profiling an Entire Session

- Create file called profiler.cfg with 3 lines:

```
-OUTFILE /tmp/profiler.out
-LISTINGS /tmp
-DESCRIBE someDescription
```

- Add –profile to session startup:

```
mpro dbName -p start.p -profile profiler.cfg
```

- Run normally.
- Terminate cleanly & analyze the output.
- Multiple gigabytes of data may be generated very quickly!!!

Profiler Tooling

zprof_on.p - initializes profiling
zprof_off.p - ends profiling and writes the oputput file
zprof_topx.p - produces a report on the results
zprof_flush.p - flushes profiling output, this is optional but necessary if your program does not terminate cleanly
zprof_check.p - checks for a basename.zprof "flag" file and starts profiling if it exists, stops profiling if it does not

to use:

```
run lib/zprof_on.p ( "baseName", "description", no ).
run interestingStuff.p
run lib/zprof_off.p.
```

This will create a single basename*.prf file that can be loaded into a profiler analysis tool such as zprof_topx.p

```
proenv>  pro -p  lib/zprof_topx.p  -param tmp/basename.2016.12.18.11.12.14
```

Sample Profiling Output

Profiler: Top 20 Results

Description: ProTop3 Execution Profile [00:00:23]
Session Total Execution Time 00:00:06
Line 0 = initialization, line -1 = cleanup

Top 20 Lines: Total Execution Time

Program/Class	Line	Time	Avg Time	Calls	Internal Procedure/Method
dc/dashboard.p	3105	3.702797	0.000001	4000004	mon-update
dc/dashboard.p	0	2.097173	0.524293	4	mon-update
dc/dashboard.p	3106	0.497054	0.000000	4000000	mon-update
dc/dashboard.p	3087	0.393956	0.004061	97	mon-update
lib/dynscreen.p	1238	0.025957	0.006489	4	dynScreenUpdate
ssg/sausage00.p	1949	0.024891	0.000007	3472	dataSet2JSON
lib/dynscreen.p	936	0.021612	0.000027	800	dynViewerUpdate
ssg/sausage00.p	1752	0.020572	0.000007	3084	scanDataSet
ssg/sausage00.p	1920	0.019844	0.000005	4252	dataSet2JSON

DEBUG Listing

```
3101  if pt_bogomips > 0 /* and bogoLoop > 1 */ then
3102    do:
3103
3104      estart = etime.
3105      do i = 1 to pt_bogomips:   /* 100,000 should take 20ms to 30ms on c.2010 HW    */
3106      end.
3107
3108      tt_Dashboard.bogoMIPS = ( 1000 / ( etime - estart )) * ( pt_bogomips /  1000000 ).
3109
3110      if tt_Dashboard.bogoMIPS > 0 and tt_Dashboard.bogoMIPS <> ? then
3111        do:
3112          maxBogo = max( maxBogo, tt_Dashboard.bogoMIPS ).
3113          tt_Dashboard.bogoMipPct = ( tt_Dashboard.bogoMIPS / maxBogo ) * 100.
3114        end.
3115
3116    end.
```

About Line Numbers

- Many tools use DEBUG LIST line numbers.
- This listing expands all include files and pre-processor statements.
- COMPILE “program.p” DEBUG-LIST “program.dbg”.
- Line numbers can vary if things have changed since code was deployed!
  - It is helpful to create and deploy .dbg files whenever you create and deploy r-code


easy

Caveat!

```
define variable i as integer no-undo.

assign
    profiler:enabled = yes
    profiler:profiling = yes
.

do i = 1 to 1000000:
end.

i = 0.
do while i < 1000000:
    i = i + 1.
end.

i = 0. do while i < 1000000: i = i + 1. end.

assign
    profiler:enabled = no
    profiler:profiling = no.

profiler:write-data().
```

Caveat!

```
1 02/28/2007 "Generic" 07:55:03 "tom"
.
2 "profile.p" "" 63126
.
0 0 2 1
.
0 0 1 0.000000 30.935828
2 11 1 0.000002 0.000002
2 8 1000001 4.607678 4.607678
2 9 1000000 1.719586 1.719586
2 14 1000000 1.501487 1.501487
2 12 1000001 3.013981 3.013981
2 13 1000000 3.032433 3.032433
2 16 1 0.000003 0.000003
.
.
.
.
```

Note 1: There are 2 formats – the 1st character specifies the version

Note 2: The input file is in “dot-d format”, a “.” on a line causes a 4GL LEAVE.

User Table and Index Statistics

User Table and Index Statistics

- Aggregate Table and Index stats were introduced in 8.3
- That was such a great feature that user level stats were introduced in OE 10!
- Now you can see how much of your database activity is from a given user.
- This is run-time behavior – not static, compile time analysis of index selection; IOW, what **really** happens vs what “should” happen.

Enabling User Table and Index Statistics

-basetable default = 1

-baseindex default = 1

-tablerangesize default = 50 select count(*) from _file

-indexrangesize default = 50 select count(*) from _index

Metaschema and VST tables have negative _file-num, SYS* start after 32000 – User tables are generally from 1 to 10000.

Index _idx-num has no pattern.

Gathering User Table & Index Statistics

```
run lib/usertablestats.p persistent.

for each dictdb.order no-lock:
end.

{lib/userstats.i}
run getUStats ( output table tt_usrTblInfo by-reference,
  output table tt_usrIdxInfo by-reference ).

for each tt_usrTblInfo by tt_usrTblInfo.tblRd descending:
  display tblName tblRd tblCr tblUp tblDl with 5 down.
end.

for each tt_usrIdxInfo by tt_usrIdxInfo.idxRd descending:
  display idxName idxRd idxCr idxDl with 5 down.
end.
```

Top 5 User Tables & Indexes

Top 5 Tables Used by My Session

tblName	tblRd	tblCr	tblUp	tblDl
Order-Line	2,619	0	0	0
Customer	332	0	0	0
Order	207	0	0	0
Item	165	0	0	0
Salesrep	9	0	0	0

Top 5 Indexes Used by My Session

idxName	idxRd	idxCr	idxDl
Order-Line.order-line	2,622	0	0
_Field._Field-Name	2,191	0	0
_Index._File/Index	873	0	0
_File._File-Name	577	0	0
Customer.Cust-Num	336	0	0

Caution

- BLOB and CLOB field activity is misreported!
- It will be recorded as activity on tables that have the same “table Id” as the “LOB Id” (will be fixed in OE12).
- Memory use:

  (-n + -Mn + 1) * tablerangesize * 32

  (-n + -Mn + 1) * indexrangesize * 40

Temp Table Statistics

Temp Table Statistics

- The trend is very clear – applications are increasing TT usage!
- Temp-tables and ProDataSets are vital components of modern applications
- Programmers have very little insight into how the temp tables in their code are behaving
- Temp Table Statistics were introduced in OE11

Progress.Database.TempTableInfo

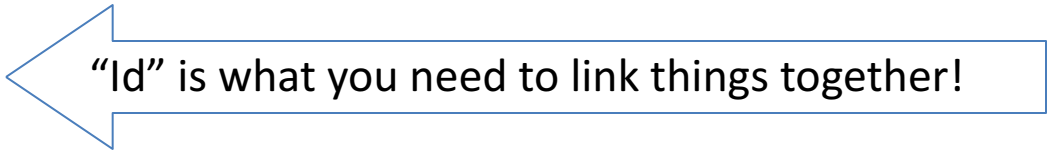
- ArchiveIndexStatistics
- ArchiveTableStatistics
- TempTableCount
- TempTablePeak

- GetTableInfoByPosition()

- GetTableInfoByID()
- GetTableStatHistoryHandle()

- GetIndexInfoByID()
- GetIndexStatHistoryHandle()

- GetVSTHandle()



- -ttbaseindex 1
- -ttbasetable 1
- -ttindexrangesize 1000
- -tttablerangesize 1000

White Star Software

Aggregate Temp-Table Info

```
┌──────────────────────────── Temp-Table Info ─────────────────────────────┐
│                                                                          │
│                 /home/pt3dev/tmp/DBI-9950762889q2d8B                     │
│                                                                          │
│     1048576 DBI File Size                  84 current temp-tables        │
│        1KB TT DB Block Size                 5 archived                   │
│        1288 TT DB Total Blocks            125 peak                       │
│         193 TT DB Empty Blocks            275 tt indexes                 │
│           2 TT DB Free Blocks            1669 total current records      │
│           0 TT DB RM Free Blocks       109831 total current bytes        │
│                                                                          │
│                           99.53% tt hit ratio                            │
│                                                                          │
│                         3225 tt rec create                               │
│                        34660 tt rec read                                 │
│                         3032 tt rec update                               │
│                           85 tt rec delete                               │
│                        96186 tt rec log rd                               │
│                          453 tt rec os rd                                │
│                         1046 tt rec os wr                                │
│                         5376 tt TRX                                      │
│                           64 tt Undos                                    │
│                                                                          │
│ ──────────────────────────────────────────────────────────────────────── │
│                              <OK> <Help>                                 │
└──────────────────────────────────────────────────────────────────────────┘
```

Detailed Temp-Table Info

TT Name	Procedure Name	Bytes	Records	Create	Read	Update	Del	OSRd
tt_tbl	protop.p	5863	184	184	17145	9		3
tt_tbl.xid-idx	protop.p			185	17801			
tt_idx	protop.p	10650	201	201	416	32		4
tt_idx.xid-idx	protop.p			202	744			
tt_screenElement	lib/dynscreen.p	34254	408	408	825	165		34
tt_screenElement.scrFrame	lib/dynscreen.p			418	1409			
tt_screenElement.elNm_frNm_elH	lib/dynscreen.p			418	407			
tt_browseColumnList	lib/dynscreen.p	2701	65	65	989	37		4
tt_browseColumnList.brwCol	lib/dynscreen.p			65	468			
tt_browseColumnList.brwHdl	lib/dynscreen.p			102	734			

Embedded Nonsense

Embedding Instrumentation

- Do not disturb the user!
- Make it easy to enable or disable remotely and IN PRODUCTION
- Gather detailed data only when you need it.
- Simple and immediately useful output.

Embedding in a “Headless” Process

```
output stream outStrm to value( flgFileName ).

do while true:

  file-info:file-name = flgFileName.            /* check (frequently)...                        */
  if file-info:full-pathname = ? then leave.    /* if the flag disappears then gracefully exit  */

  file-info:file-name = dbgFileName.
  if file-info:full-pathname <> ? And
   ( file-info:file-mod-date <> dbgChgDate or file-info:file-mod-time <> dbgChgTime ) then
    do:
      dbgChgDate = file-info:file-mod-date.     /* remember when we last looked!                */
      dbgChgTime = file-info:file-mod-time.
      input stream inStrm from value( file-info:full-pathname ).
      import stream inStrm dbgMode.             /* read the new value of “dbgMode”              */
      input stream inStrm close.
      if dbgMode = “profiler” then run lib/zprof_on.p.
    end.

  /* do the work… */

end.
```

White Star Software

Embedding in a “Headless” Process

```
do while true:
  run lib/zprof_check( "baseName", "description", yes ).
  run interestingStuff.p
  pause 10 no-message.
end.
```

/* This approach allows you to enable and disable profiling externally -- just create baseName.zprof to
 * enable profiling and remove it to disable profiling. with the zFlush flag set to "yes" each iteration
 * of the loop will create a basename.YY.MM.DD.HH.MM.SS.prf output file. */

Embedding in a “Headless” Process

```
/* zprof_check.p -- simple stand-alone non-persistent profiler control
 *
 * run lib/zprof_check.p ( "baseName", "description", zFlush ).
 *
 * check for baseName.zprof in session:temp-directory -- if it exists
 * start profiling, if it does not then end profiling
 * if baseName is blank or unknown then check for "zprof.zprof”
 */

flagfile = ( if zBaseName = "" or zBaseName = ? then "zprof” else zBaseName ).

file-info:file-name = session:temp-directory + "/" + flagFile + ".zprof".
if file-info:full-pathname = ? then
  run lib/zprof_off.p.
 else
  run lib/zprof_on.p ( zBaseName, zDescription, zFlush ).
```

Resources!

- All provided as part of protop http://protop.wss.com

  Profiler: lib/zprof*.p use the “y” command (lower case)

  TT Stats: lib/ttinfo.p use the “0” command (zero)

  User Stats: lib/usertablestats.p use the “Y” command (upper case)

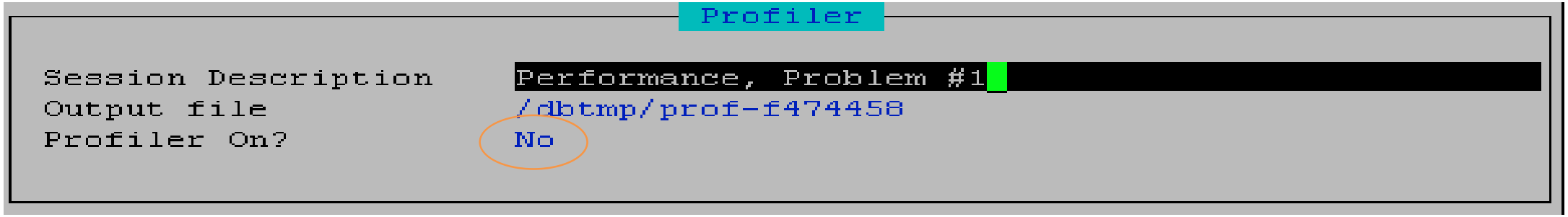
- Cool Kids:
  - https://github.com/consultingwerk/TableStatistics

Questions?

Thank You!

Demo!

<Escape>-P Profiler Tool



- Dynamically capture line by line execution time at the point where issues occur.
- Send output to a user-defined destination.
- <Esc>P to configure and enable. (Don’t forget “Yes”.)
- <Esc>P again to complete the capture and disable.

Profiler Example
A Calculation Bottleneck?

$$\pi = 4 * \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \ldots\right)$$

Profiler Example
A Calculation Bottleneck?

```
function piterm returns decimal ( input n as integer ).
  return ( 1.0 / (( n * 2 ) + 1 )).
end.

do while abs( newpi - oldpi ) > precision:
  oldpi = newpi.
  i = i + 1.
  if i modulo 2 = 0 then
    pi = pi + piterm( i ).
   else
    pi = pi - piterm( i ).
  newpi = ( 4.0 * pi ).
  display newpi oldpi.
end.
```