ProDelphi

Reviewed by Craig Murphy

Those of you who have taken a look at the other goodies on the Delphi 6 and 7 CDs may have noticed a code profiling tool on the Companion CD. However, if you are anything like me, you probably installed Delphi and went back to work, ignoring all the additional goodies that are 'in the box'!

ProDelphi is a code profiling tool for the Win32 and .NET versions of Delphi, ie versions 2 through to 8. It has a lengthy history stretching back as far as 1998, and came about after work on Turbo Profiler ceased. This review used ProDelphi 15.4, Delphi 6 and Windows XP.

If your users are complaining about application performance, ProDelphi could be the tool for you: its authors have used it to increase application runtime performance by 50%. In other words, the amount of time the application spent achieving its end goal was reduced by 50%! Basically, it ran a lot faster. If this is for you, or rather for your users, then please read on.

Profiling Basics

There are three basic types of profiling: sampling, source instrumentation and machine code instrumentation.

Sampling involves taking regular snapshots where the profiler works out which method is currently being executed. It is possible that some methods will be ignored because the interval between snapshots is too long. Equally, the precise execution time for a method cannot be determined, only a rough figure.

Instrumentation involves decorating the source code with calls to the profiler. It is arguably intrusive, but does provide us with accurate execution times, and it covers all methods unless told otherwise. The cost of calling the profiling functions, typically at the start and end of a method call, is consistent and can be measured. This is known as 'API overhead'.

Machine code instrumentation leaves the source code intact but decorates/augments the machine code that the compiler creates. Whilst you may believe that this method offers the most accurate and most precise means of timing method execution, because of the ways processor caches work the imprecision of this method is emphasised by the fact that the profiling code may have to be loaded into the processor's cache. Under these circumstances it is very difficult to accurately derive the API overhead.

There is a good graphical description of each type of profiling at the ProDelphi website: www.prodelphi.de/sample.htm.

ProDelphi is a tool that instruments or decorates code with profiling instructions. Its authors have gone to exceptional lengths to ensure that the API overhead does not skew the profiler's results.

ProDelphi's Modes Of Operation

ProDelphi offers three types of profiling: call count, function and emulation.

Call count profiling gathers usage statistics relating to the number of calls a method receives. Function profiling works out the average runtime (or length of execution) for each method.

Emulation profiling allows us to simulate a slower computer. This is useful if your development machines are all top of the range, high spec, high performance, etc. Users often have considerably less powerful machines, being able to 'dumb down' your application to their machine's pace will help you find (and remove!) bottlenecks that might cause your users frustration.

ProDelphi actually performs each of these types of profiling during each profiling session.

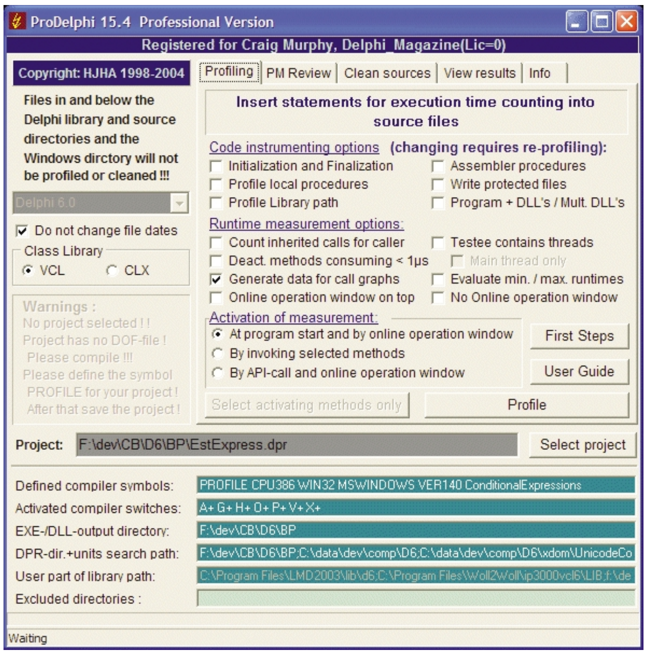
For reasons of performance, ProDelphi does not offer a line-by-line profiling mechanism. Instead, for coverage profiling, ProDelphi's authors recommend a product called Discover by Cyamon Software *[Look out for a review of Discover next month! Ed]*.

Profiling By Instrumentation

Asking ProDelphi to profile an application is an easy task. The first thing we must do is define `PROFILE` using `Project | Options | Directories/Conditionals`. Our second step is to load ProDelphi, either via the `Start` menu or via the IDE integration: `Tools | ProDelphi`.

Figure 1 presents the ProDelphi opening screen. ProDelphi's user interface is busy, to say the least, but it is easy to use and does its job well.

I chose to profile my largest (and oldest) application. EstExpress is an estimating application comprising nearly 200,000 lines of code,


ProDelphi 15.4 Professional Version

Registered for Craig Murphy, Delphi_Magazine(Lic=0)

Copyright: HJHA 1998-2004

Profiling | PM Review | Clean sources | View results | Info

Files in and below the Delphi library and source directories and the Windows dirctory will not be profiled or cleaned !!!

Delphi 6.0

[✓] Do not change file dates

Class Library: [✓ VCL] CLX

Warnings :
No project selected ! !
Project has no DOF-file !
Please compile !!!
Please define the symbol PROFILE for your project !
After that save the project !

Insert statements for execution time counting into source files

Code instrumenting options (changing requires re-profiling):
[ ] Initialization and Finalization
[ ] Assembler procedures
[ ] Profile local procedures
[ ] Write protected files
[ ] Profile Library path
[ ] Program + DLL's / Mult. DLL's

Runtime measurement options:
[ ] Count inherited calls for caller
[ ] Testee contains threads
[ ] Deact. methods consuming < 1µs
[ ] Main thread only
[✓] Generate data for call graphs
[ ] Evaluate min. / max. runtimes
[ ] Online operation window on top
[ ] No Online operation window

Activation of measurement:
[✓ At program start and by online operation window]
By invoking selected methods
By API-call and online operation window

First Steps | User Guide | Select activating methods only | Profile

Project: F:\dev\CB\D6\BP\EstExpress.dpr | Select project

Defined compiler symbols: PROFILE CPU386 WIN32 MSWINDOWS VER140 ConditionalExpressions
Activated compiler switches: A+ G+ H+ O+ P+ V+ X+
EXE-/DLL-output directory: F:\dev\CB\D6\BP
DPR-dir.+units search path: F:\dev\CB\D6\BP;C:\data\dev\comp\D6;C:\data\dev\comp\D6\xdom\UnicodeCo
User part of library path: C:\Program Files\LMD2003\lib\d6;C:\Program Files\Woll2Woll\ip3000vcl6\LIB;f:\de
Excluded directories :

Waiting

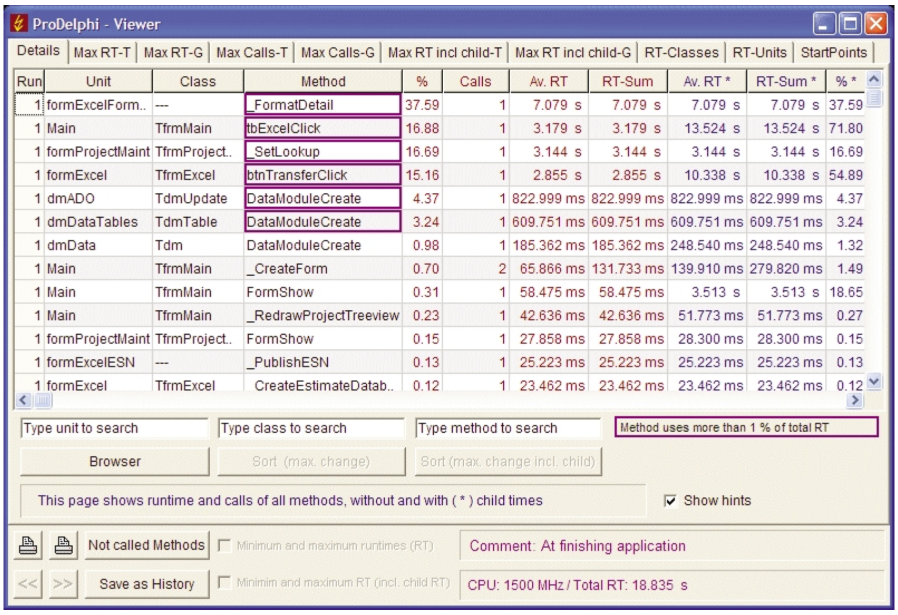
➤ *Figure 1: ProDelphi's opening screen.*

100 forms and two InterBase databases. One of the things that my users complain about is the speed of reporting: it works, it's flexible, it's customisable, but they want speed!

After defining `PROFILE`, it is necessary to click on Figure 1's `Select Project` button. ProDelphi then provides us with some information about the compiler symbols, switches, output directory, etc. Clicking on the `Profile` button then instructs ProDelphi to traverse the selected project, augmenting it, or 'instrumenting' it, with profiling code. I chose to tick the `Do not change file dates` checkbox simply to stop Delphi wanting to reload .pas files after they had been augmented.

On the whole, the augmentation process worked seamlessly on vanilla .pas files. I did encounter an occasion where it did not leave my .dpr file in a state that it would compile, but this issue is due to be resolved in a future version. My .dpr file is a little more convoluted than normal: it has a one or two `if` conditions wrapping up `Application.Run`. It was not a showstopper: a little bit of hand-crafting and it compiled with profiling. I discovered later that if I inserted redundant `BEGIN..END` states around my `IF` conditions, ProDelphi was much happier.

➤ *Figure 2: It's easy to see where the run-time is being spent.*



Run	Unit	Class	Method	%	Calls	Av. RT	RT-Sum	Av. RT *	RT-Sum *	% *
1	formExcelForm..	---	_FormatDetail	37.59	1	7.079 s	7.079 s	7.079 s	7.079 s	37.59
1	Main	TfrmMain	tbExcelClick	16.88	1	3.179 s	3.179 s	13.524 s	13.524 s	71.80
1	formProjectMaint	TfrmProject..	_SetLookup	16.69	1	3.144 s	3.144 s	3.144 s	3.144 s	16.69
1	formExcel	TfrmExcel	btnTransferClick	15.16	1	2.855 s	2.855 s	10.338 s	10.338 s	54.89
1	dmADO	TdmUpdate	DataModuleCreate	4.37	1	822.999 ms	822.999 ms	822.999 ms	822.999 ms	4.37
1	dmDataTables	TdmTable	DataModuleCreate	3.24	1	609.751 ms	609.751 ms	609.751 ms	609.751 ms	3.24
1	dmData	Tdm	DataModuleCreate	0.98	1	185.362 ms	185.362 ms	248.540 ms	248.540 ms	1.32
1	Main	TfrmMain	_CreateForm	0.70	2	65.866 ms	131.733 ms	139.910 ms	279.820 ms	1.49
1	Main	TfrmMain	FormShow	0.31	1	58.475 ms	58.475 ms	3.513 s	3.513 s	18.65
1	Main	TfrmMain	_RedrawProjectTreeview	0.23	1	42.636 ms	42.636 ms	51.773 ms	51.773 ms	0.27
1	formProjectMaint	TfrmProject..	FormShow	0.15	1	27.858 ms	27.858 ms	28.300 ms	28.300 ms	0.15
1	formExcelESN	---	_PublishESN	0.13	1	25.223 ms	25.223 ms	25.223 ms	25.223 ms	0.13
1	formExcel	TfrmExcel	CreateEstimateDatab..	0.12	1	23.462 ms	23.462 ms	23.462 ms	23.462 ms	0.12

Method uses more than 1 % of total RT

This page shows runtime and calls of all methods, without and with (*) child times

[✓] Show hints

[ ] Minimum and maximum runtimes (RT)

[ ] Minimim and maximum RT (incl. child RT)

Comment: At finishing application

CPU: 1500 MHz / Total RT: 18.835 s

```
procedure TfrmMain.Change(obState : TObsState);
var
  Obs: TSubjectObserver;
  I: Integer;
begin
{$IFDEF PROFILE}
  asm DW 310FH; call Profint.ProfStop; end; Try; asm mov edx,1225;
  mov eax,self; call Profint.ProfEnter; mov ecx,eax; DW 310FH;
  add[ecx].0,eax; adc[ecx].4,edx; end;
{$ENDIF}
  for I := 0 to FObservers.Count - 1 do begin
    Obs := FObservers[I];
    Obs.State := obState;
    if Obs.Enabled then
      Obs.Change;
  end;
{$IFDEF PROFILE}finally; asm DW 310FH; mov ecx,1225; call Profint.ProfExit;
  mov ecx,eax; DW 310FH; add[ecx].0,eax; adc[ecx].4,edx; end; end; {$ENDIF}
end;
```

➤ *Listing 1: Instrumented code.*

After an application has been augmented with profiling code, we need to re-compile it then run it. Whilst the application is running, ProDelphi is monitoring which procedures and functions are invoked, how long each one takes to run, and in what order they are called. This information allows the creation of output like that shown in Figures 2 and 3, both of which we will discuss later in this review.

Listing 1 presents an instrumented procedure from the EstExpress application: the instrumentation code is shown in red. By augmenting the .pas files in a project, ProDelphi is able to call its own library functions that record how long each procedure or function takes to execute and how often it is called.

It is possible to include standard Delphi comments that instruct ProDelphi to ignore specific chunks of code. These comments remain in the source code even after ProDelphi's profiling code has been 'cleaned' or removed. Clicking on Figure 1's `Clean sources` tab gives us the chance to return our augmented source back to its former glory, without any profiling code added.

Granularity

ProDelphi's primary unit of measurement is CPU cycles, thus the smallest measurable unit is 1 CPU cycle. On a 1,000Mhz processor this means that the smallest measurable duration is 1 nanosecond.

Figure 2 presents ProDelphi's internal viewer. It shows which methods consumed most processor time (or run time). As you can see, the EstExpress report routine is measured using seconds and milliseconds, not nanoseconds! The columns `Run`, `Unit`, `Class`, `%`, `RT-Sum`, etc, are clickable and sort the columns just as you would expect.

If a particular method is called more than once, the `Av.RT` column would indicate the average runtime consumed and the `RT-Sum` column would indicate the total of all the calls put together.

CallGraphs

Clicking on any of the methods shown in Figure 2 opens a CallGraph window. Looking back to Figure 1, it is necessary to ask ProDelphi to generate the data for call graphs by ticking the appropriate checkbox. Figure 3 presents the CallGraph for the EstExpress reporting routine. If it was not obvious from Figure 2 (which it was!), the layout of the CallGraph makes it very clear that _FormatDetail is the culprit!

Items closer to the centre of the CallGraph take more runtime than those nearer the periphery. The CallGraph viewer is a really nice piece of work: it is fully clickable, allowing the rapid drill-down into a particular method. Often a method is portrayed as being very slow: using the drill-down features of the CallGraph viewer we can home in on the specific slow child methods and classes or sub-routines. After all, like most things, it is easier to optimise smaller pieces of code, than to try and optimise a larger chunk.

Incidentally, the offending _FormatDetail function relies on a COM connection to Microsoft Excel and that is why the performance hit is noticeable. Experience has taught me that where such a link between a Delphi application and Excel is required, it is better to write an Excel macro that performs the same task as the Delphi code: it is significantly faster than the COM equivalent.

Download And Installation

ProDelphi is available in two flavours: one is for Win32 versions of Delphi (2 through to 7); the other is for Delphi 8. Each download is roughly 1.2Mb in size.

Costings And Availability

ProDelphi has two modes of operation: Freeware and Professional. Freeware mode limits the number of procedures and functions that ProDelphi will profile to 20. The Freeware mode also ignores any assembler routines.

Registering ProDelphi provides access to the Professional mode: it allows profiling of up to 64,000 functions and procedures, and opens up profiling of assembler routines. The Professional mode leaves the file's date/time stamp unchanged during the profiling process. The Professional version can be purchased via ShareIt for €49.50.

For a tool that could help you improve your application's runtime performance by significant percentages, it is an absolute bargain. Application performance is something many users strive for: imagine being able to shave up to 50% off the run time simply by performing a handful of carefully selected optimisations? ProDelphi can help you achieve such a performance gain, and for very little financial outlay.

For your €49.50 you will receive free updates for that platform: if you own ProDelphi for Win32, all updates for that version are free. However, if you wish to use ProDelphi with Delphi 8, you will need to purchase another copy: it is a different platform so warrants the additional licence cost.

Help And Tutorials

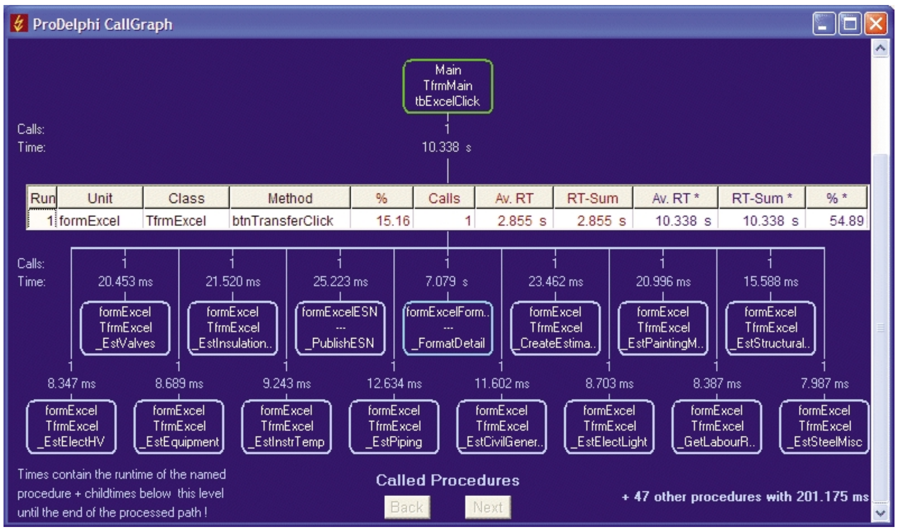
ProDelphi's documentation, both printed and on-line, is excellent and is very comprehensive. Some parts of the documentation contain spelling mistakes and some interesting use of the English language. However, given the cost and the quality of the product, and the fact its authors write and speak better English than I do German, I would not let this put you off purchasing the Professional version.

Accompanying ProDelphi is a 41-page User Guide. This provides a bit of the history behind ProDelphi and profiling, a step-by-step example, and explanations of the file formats used. ProDelphi creates up to five files during the profiling process. Whilst it is possible to use the built-in ProDelphi viewers, having access to the file formats lets us import the files into other applications, such as a database, for example.

Conclusions

ProDelphi is certainly a competent product: it does what it says it will do and that is to help optimise your application's runtime performance. Within five minutes of installing ProDelphi and asking it to profile my largest application, it had correctly identified the areas that I should be concentrating my efforts on. The CallGraph viewer is an amazing tool in its own right: being able to visualise code structure and the run time it consumes, and being able to drill-down into the code at the same a boon. ProDelphi: €49.50 well spent.

➤ *Figure 3: ProDelphi can generate a CallGraph.*


ProDelphi CallGraph
Main TfrmMain tbExcelClick
Calls: 1
Time: 10.338 s
formExcel TfrmExcel _EstValves — 1, 20.453 ms
formExcel TfrmExcel _EstInsulation.. — 1, 21.520 ms
formExcelESN --- _PublishESN — 1, 25.223 ms
formExcelForm.. --- _FormatDetail — 1, 7.079 s
formExcel TfrmExcel _CreateEstima.. — 1, 23.462 ms
formExcel TfrmExcel _EstPaintingM.. — 1, 20.996 ms
formExcel TfrmExcel _EstStructural.. — 1, 15.588 ms
formExcel TfrmExcel _EstElectHV — 1, 8.347 ms
formExcel TfrmExcel _EstEquipment — 1, 8.689 ms
formExcel TfrmExcel _EstInstrTemp — 1, 9.243 ms
formExcel TfrmExcel _EstPiping — 1, 12.634 ms
formExcel TfrmExcel _EstCivilGener.. — 1, 11.602 ms
formExcel TfrmExcel _EstElectLight — 1, 8.703 ms
formExcel TfrmExcel _GetLabourR.. — 1, 8.387 ms
formExcel TfrmExcel _EstSteelMisc — 1, 7.987 ms
Called Procedures
Back
Next

Run	Unit	Class	Method	%	Calls	Av. RT	RT-Sum	Av. RT *	RT-Sum *	% *
1	formExcel	TfrmExcel	btnTransferClick	15.16	1	2.855 s	2.855 s	10.338 s	10.338 s	54.89

Times contain the runtime of the named procedure + childtimes below this level until the end of the processed path !

+ 47 other procedures with 201.175 ms

Craig is an author, developer, speaker, project manager and Certified ScrumMaster. He specialises in all things XML, particularly SOAP and XSLT. Craig is also evangelical about Test-Driven Development and Extreme Programming. He can be reached via email at: tdm@craigmurphy.com or via his website at www.craigmurphy.com

Resources

ProDelphi is available here:
www.prodelphi.de

Discover by Cyamon Software is available here:
www.cyamon.com