

AProf

COLLABORATORS

	<i>TITLE :</i> AProf	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY		March 28, 2025
<i>SIGNATURE</i>		

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	AProf	1
1.1	AProf:guide	1
1.2	AProf:What is a profiler	1
1.3	AProf: System requirements	2
1.4	AProf:Installation procedure	2
1.5	AProf: Starting the profiler	2
1.6	AProf:User interface	3
1.7	AProf:Menu: Files	3
1.8	AProf:Menu: Action	3
1.9	AProf:Menu: Data	4
1.10	AProf:Menu: Move	4
1.11	AProf:Menu: Misc	4
1.12	AProf: Profiler mode: Separate/Combined	5
1.13	AProf: Cycle: Percentual/Millisecond timevalues	5
1.14	AProf: XTime	5
1.15	AProf: Configurability	6
1.16	AProf: Notes	6
1.17	AProf: Caveats	7
1.18	AProf: StripB	8
1.19	AProf: Tested systems	8
1.20	AProf: Tested systems: Amiga E	9
1.21	AProf: Tested systems: Aztec C	9
1.22	AProf: Tested systems: SAS C	9
1.23	AProf: Tested systems: Maxon C++	9
1.24	AProf: Tested systems: DICE C	10
1.25	AProf: Tested systems: GNU C/C++	10
1.26	AProf: Tested systems: Assemblers	10
1.27	AProf: Tested systems: Other...	10
1.28	AProf: Feedback	12
1.29	AProf needs you...	12

1.30 AProf: Copyright	13
1.31 AProf: Changes	13
1.32 AProf: Question and Answer	14
1.33 AProf: Preferences dialog	14
1.34 AProf: Rexx example	15
1.35 AProf: Exec dialog	15

Chapter 1

AProf

1.1 AProf.guide

Documentation for Amiga Profiler V3.30

© 1993,94 Michael G. Binz

What is a profiler?

System requirements

Installation procedure

Starting the profiler

User interface

Configurability

Notes

Tested Systems

Caveats

StripB

Feedback

AProf needs you...

Copyright

Changes

Q+A

1.2 AProf:What is a profiler

A profiler is a development tool supporting the developer in code optimization. A specified program is executed by the profiler which collects several informations (ie. call counts, time values) while execution. This data can be used to locate the so-called hot spots in your code - functions where most of the execution time is spent and optimization will gain most effect.

The following articles supply further information on profilers:

- o Dr. Dobb's Journal, January 1993
Joseph Newcomer, 'Profiling for Performance', pp. 80
- o Dr. Dobb's Journal, November 1993
Michael R. Dunlavey, 'Performance Tuning: Slugging it out', pp. 18

1.3 AProf: System requirements

- o Hardware requirements

The profiler needs at least Workbench/Kickstart 2.04 and a minimum of 500K available memory (depends on profilees sizes).

It has been tested successfully with Amiga 500, 2000, 3000, 4000 and OS versions 2.04, 2.1, 3.0.

- o Software requirements

Your compiler must be able to create Amiga symbol hunks. Code symbols included in this hunks should only address the starting addresses of functions in your code. No intermediate jump labels must be defined. (See ["Clean symbol tables" link frageantwort](#)), [" ← Tested compilers" link l-systeme](#) }

1.4 AProf:Installation procedure

Copy all files included in AProf3_30.lha into a directory and add this directory to your PATH.

Do not delete AProf's icon (AProf.info), since configuration data is located in this file (See Configurability).

1.5 AProf: Starting the profiler

From CLI the profiler is started with

```
1.> AProf [application]
```

The profiler starts up, displays its user interface, loads the symbol tables from the file 'application' if specified and waits for your actions.

From Workbench, doubleclick the AProf icon.

If no filename was specified or if started from workbench, you can load a file from the menu ["\[Files/Open\]" link m-files](#) }.

See also: ["User interface" link l-ui](#) }

1.6 AProf:User interface

Title: Name of loaded profilee

Menus: @ { " Files " link m-files } @ { " Action " link m-action } @ { " Data " ↔
link m-data } @ { " Move " link m-move } @ { " Misc " link m-misc }

Displays: @ { " Mode " link gad-inex } @ { " Time unit " link gad-percmil } @ { " ↔
Sort " link gad-sort } @ { " Time " link gad-xtime }

Buttons: [Shortcuts for often needed menu entrys]

Description of table entrys:

Function	HitCnt	Per Call	Over	Min/Max
Symbol names*				
Call count**				
Average execution time per call				
Overall execution time for that function				
				Minimum/maximum execution time per call

* The symbol *ENTRY*, if displayed, is generated by AProf if the symbol table of the profilee contains no symbol sitting at the very first executed address of its seglist.

** A call count value of -1 is shown for symbols being no function entrys. Safe profiling must be active.

[Status and error messages are displayed in the bottom window border]

1.7 AProf:Menu: Files

Menu: Files

Reading and writing profiler data.

Open: Read a new file (also accessable via button)

Save: Save report to file with same basename and suffix .pro

Save as: Save report to selectable file

Reset: Reset all timing values

Print: Print report

Exit: Leave the program

1.8 AProf:Menu: Action

Menu: Action

Start the profiling process.

Start: Starts the profile run. If you restart a profilee, then times are added.

If you want to run a profilee several times without adding the times and hit counts, you must reset the timer and hitcounts with @{" Files/Reset " link m-files }.
(also accessible via button row)

1.9 AProf:Menu: Data

Menu: Data

Runtime configuration and execution control.

Exec details:

Provide a command line and stack size for the profilee

See: @{"Exec dialog" link dialog-exec }

Preferences:

Configure the profiler.

See: @{"Preferences dialog" link dialog-preferences }, @{"Configurability" link l-config } ←

1.10 AProf:Menu: Move

Menu: Move

Moving in the display area.

Find: Enter a search string and start search

Find next: Search for next occurrence

Top: Go to top of symbol table

Bottom: Go to bottom of symbol table

Page up/down: One page up/down

1.11 AProf:Menu: Misc

Menu: Misc

Other functions

Help:

Start the AmigaGuide(TM) hypertext help system (if available)

Refresh Window:

Redisplay symbol table

About:

Display version and copyright information

1.12 AProf: Profiler mode: Separate/Combined

Separate/Combined: (Mode)

Displays how function timing values are accumulated. If combined is selected, the time value for a function includes the times of all functions that are called by this function. Separate excludes the times of called functions.

Note that if your code includes recursive procedure you must use separate mode. If combined is used, the time for the recursive procedure is wrong.

Example:

```
int main( void )
{
    foo( 2 ); /* foo needs 2secs execution time */

    bar( 3 ); /* bar needs 3secs execution time */

    ...burn 1sec proc. time in main()...

    return 0;
}
```

If 'Separate' is selected, the profile list will look like this:

Symbol	HitCnt	Per Call	
main	1	1000	The timing values are updated for each funktion separate.
foo	1	2000	
bar	1	3000	

Now if 'Combined' is selected:

Symbol	HitCnt	Per Call	
main	1	6000	The timing values for foo() and bar() are added to main(), since main() calls both.
foo	1	2000	
bar	1	3000	

1.13 AProf: Cycle: Percentual/Millisecond timevalues

Millisecond/Percentual time units: (Units)

Displays the units used for for timing values. If percentual is active, times shown are fractions of overall execution time in percent.

1.14 AProf: XTime

Overall execution time of profilee. Units are seconds or milliseconds depending on execution time. Times longer than 1000 ms are displayed in seconds (s), below in milliseconds (ms).

1.15 AProf: Configurability

All configurable items of AProf must be set as ToolTypes in AProf's .info file, which must reside in the same directory as the profiler.

```
+-----+
|      Never delete AProf.info! All configuration data will be      |
|                               written to this file!                |
+-----+
```

Beginning with version 3.30 one can save AProf's current settings with the 'Save' button in 'Preferences'. It is recommended to use this instead of setting the ToolTypes manually.

- o WINDIM=(left/top/width/height)
Use this to set size and position for the profiler window.
- o TUNITS=(PERCENTUAL | MILLISECOND)
Default time units.
- o PMODE=(COMBINED|SEPARATE)
Profiling mode
- o PATTERN=(AmigaPattern)
Amiga pattern for symbols to hide.
Look in your Amiga User Manual for a description of pattern syntax
- o SORT=(NONE | NAME | HIT | AVERAGE | OVER)
Specify the sort order you prefer.
- o SAFE=(TRUE | FALSE)
Safe profiling on or off

1.16 AProf: Notes

Implementation

AProf is an active profiler. This means, hit counts and timing values are measured by employing a sophisticated breakpoint scheme.

Before a profilee is executed by AProf, all function entry points are marked with breakpoints.

After execution is started, AProf receives the breakpoint hits and places additional breakpoints at the function return points.

The address for function return points is derived from the stack. This should make it clear why AProf needs a 'clean' symbol table: There is no foolproof way to figure out the function return address if there are any function local data on top of stack.

Actually some testing can be undergone to ensure the correct values are read from the stack. This can be activated in AProf versions

> 3.22 by the checkmark 'safe profiling' in AProf preferences. Although this is not foolproof as stated, it works unexpected good with most compilers. The disadvantage of this technique is the additional time spent in the checking routines.

Profilee environment

 Whether AProf is started from CLI or workbench, the profilee's environment is always a shell. If started from wb, a shell window will be supplied for profilee IO.

ARexx port

 In the current version AProf provides no additional functions for ARexx hosts.

1.17 AProf: Caveats

Here is a list of known constructs which can result in problems if you try to profile programs including them

o Non-standard startup code

 The profilees must be able to run in the same process environment as the profiler. It's not possible to use special startup codes for detaching a program or making it resident.

o recursive procedures

 Use only '@{ "separate" link gad-inex }' profiling mode if your code includes recursive procedures. In combined mode the time for the recursive procedure is wrong. ←

o setjmp()/longjmp()

 If setjmp()/longjmp() combinations are used in the profilee, the time span between calling longjmp() and the next RTS instruction will add to the function calling longjmp().

o signal()/raise()

 Most compiler libraries contain signal()/raise() functions which are not compatible with AProf.

o CIA Timer

 CIA timers are not available for profilees.

- o Overlays

Profiling of overlaid programs is not possible.

- o Runtime limitation

Profiler timers can measure maximum time spans of about 99 mins.

- o Static functions

Functions to be measured must be in the symbol table. This is not the case for static (module local) functions in most programming environments.

- o Traphandlers

If your program uses a private trap handler, traps not handled must be propagated to the previous handler. Used traps must be allocated with AllocTrap().

Profiling is NOT possible if you are using trace traps (#9).

- o Switch- and Launchfunctions

Profilees using members tc_Switch and tc_Launch in Exec's Task structure must propagate execution to previous defined handlers.

1.18 AProf: StripB

StripB is a command line utility which can be used to remove all HUNK_SYMBOL- and HUNK_DEBUG-hunks from an Amiga executable.

Command line: StripB infile outfile

1.19 AProf: Tested systems

This is a list of systems I have tested the profiler with. If you find an error or if you have tested a system not included here, send a message please.

```
@{ " Amiga E " link amiga-e }
@{ " Aztec C " link manx-c }
@{ " DICE C " link dice-c }
@{ " GNU C/C++ " link gnu-cpp}
@{ " SAS C 6.3 " link sas-c }
@{ " Maxon C++ V1.2.1 " link maxon-cpp }
@{ " Assemblers " link assemblers }

@{ " Other systems... " link other }
```

1.20 AProf: Tested systems: Amiga E

AProf works with all versions of Amiga E >2.1b.

Problems:

Profiling of programs using exceptions leads to meaningless results

Creation of symbol hunks:

Use -s switch

1.21 AProf: Tested systems: Aztec C

Manx Aztec C V3.4 - V5.2b

Creation of symbol hunks:

Use option -w for the linker

Problems:

- o Remove symbols named '_H#[0-9]_org'
- o Don't use detach.o with programs you want to profile.
- o ANSI C functions signal() and raise() don't handle traps the way they should. (see Caveats, custom trap-handlers)

1.22 AProf: Tested systems: SAS C

SAS C ?

1.23 AProf: Tested systems: Maxon C++

Maxon C++ V1.2.1

Problems:

Code and data symbols in link libraries reside in the same hunk. According to Jens Gelhar this will change in future versions of the library. The beta version of the updated compiler writes symbol hunks as needed by Aprof so the library problem requires a simple recompile.

Compiler is adding labels named L'num' to symbol table. This must be removed (Pattern: L#[0-9]) or you must activate 'save profiling'.

Creation of symbol hunks:

For command line compiler use option -bs.

In the integrated environment use menu 'Compiler options'.

Other:

Test the procedure rexx/maxoncpp.aprof for unmangling of c++ symbol names.

1.24 AProf: Tested systems: DICE C

DICE C V2.06.21 unregistered version

Problems:

Dice generates dirty symbol tables, activate 'Safe profiling'.

Creation of symbol hunks:

Use option -s for DCC.

Information about the commercial DICE system welcome.

1.25 AProf: Tested systems: GNU C/C++

GNU gcc - 2.3.3

Problems:

Profiling programs using ixemul.library is not possible, use static linking.
Activate 'Safe profiling'.

Creation of symbol hunks:

Always created, to not create them use -s or @{" StripB " link l-stripb }, ←
which is
included in this AProf distribution.

1.26 AProf: Tested systems: Assemblers

AProf is tested with a number of assemblers (Aztec as, asm68, DevPack).
There seem to be no problems as long as the following rules are obeyed:

- o If you export code symbols other than function labels, activate 'Safe profiling' in [Data/Prefs].
- o Don't be too creative with your control flow, eg. several starting points for functions etc. Although this will not crash AProf, the results provided will be a bit random :)
- o Do not mix data and code in one hunk.
- o Do not use self modifying code.

1.27 AProf: Tested systems: Other...

Here is a little strategy to test if your system is able to cooperate with AProf.

Step 0.

First, try to find out if your system can generate executables with

Amiga symbol hunks. You should find this information in your compiler documentation.

Then after successfully creating a executable version of the program to test, first check if your program is at least stable enough to run stand-alone without crashing the machine.

Close all applications so that if a crash happens no data is lost!

Step 1.

Start AProf and load your program. Then specify in [Data/Prefs] the symbol pattern '#?' (without quotation marks). This removes the complete symbol table, which results in a reduced protocol between AProf and your program.

When this is done, close your eyes and start your program (if you have problems in finding the right keys you can keep your eyes open). If your machine crashed, then Aprof is definitely incompatible with this programming system. The only thing to do is to reboot your machine and delete AProf. If there was no problems and your program is running, take all steps needed to stop your program.

Step 2.

Next step is to remove all symbols but one, which should be generated by you. If your program contains a function 'foo' written by you then open the prefs request and insert ~(foo) as symbol pattern. After selecting 'use', this should be the only symbol displayed (it is possible, that there is an second symbol called *ENTRY* - this is used by AProf, ignore it).

Start execution. If everything works, then again stop your program or wait for termination. Now Aprof should display the function timings.

A crash is a sign for incompatibility. Forget AProf.

Step 3.

Now remove the pattern in [Data/Prefs] and activate 'Save profiling'. Then take a look at the symbol list. Remove temporary labels or line number labels.

Examples:

```
_L0001, _L0002, _L0003, _L0004, ...
_H0_org, _H0_end, _H1_org, _H1_end, ...
@0001, @0002, @0003, ...
```

Then again start execution of your program. If everything works as it should you did it. Exit your program and AProf will generate the timing report. AProf seems to be compatible with your programming system. Save the pattern created in the preceding step and use it in future profiling sessions.

If your program crashed, the hard work begins. As we saw in Step 2, your programming system basically should work with AProf. So there must be one or several data objects in your code hunks. You will

have to find and remove them.

Step 4.

Save all settings.

1.28 AProf: Feedback

Contact me under

E-Mail: michab@informatik.fh-augsburg.de

If it's not possible to use E-Mail, write to:

Michael Binz
Bahnhofstr. 11
D-86459 Gessertshausen

If you made tests with a compiler not listed in 'systems', please send me a small demo source file, the translated executable (with symbol hunk) and some information about creating symbol hunks with your compiler. If needed, send a description of problems and actions you performed to solve them, too.

1.29 AProf needs you...

... if you have access to an Amiga with MMU!

Since I'm developing AProf on an Amiga 2000 with 68000 I can't check for ENFORCER HITS! Although much care was taken to keep this nasty sort of bugs out of my code, there must be some. Anywhere.

So man, read the following!

```
if ( found_enforcer_hit() )
{
    if ( have_access_to_email() )

        Send_Micha_An_Email_Report( ); /* Really! */

    else if ( !too_lazy() )

        Send_Micha_SnailMail_Report(); /* Cool... */

    else

        GuruForever();                /* Gnah... */
}

@{ " Micha's address? " link 1-feedback }
```

1.30 AProf: Copyright

This software is freeware. You are allowed to copy, distribute and use it, as long as you don't change the software and distribute only the packed file (AProf3_30.lha).

You are not allowed to charge a fee higher than Fred Fish's for copying and distributing.

These files and their related documentation, utilities and examples are provided "AS-IS" and subject to change without notice; no warranties are made. All use is at your own risk. No liability or responsibility is assumed.

This software, the included utilities and documentation are
© 1993-94 Michael G. Binz

1.31 AProf: Changes

V 3.34

Bugs fixed:

- o Screen updates caused enforcer hits
- o Shell display of command line caused enforcer hit

Changes:

- o Console window opened when started from WB now has close gadget
- o XTime display uses [s] if [ms] value exceeds 1000

V 3.30

Future:

- o Passive (interrupt controlled) profiling mode

Known Problems:

- o AmigaGuide needs 'topaz' or maybe another fixed width font to be the system default font. If another font is used, AmigaGuide fails silently.

Changes:

- o If run from WB the profiler acts as shell for profilees
- o System default font is used
- o Added sorting functions
- o Extended configurability
- o Maximum command line length is now 256 chars
- o Special 'safe' profiling mode added
- o ARexx Port for custom symbol name unmangling

Bugs fixed:

- o Use of other fonts than topaz.8 trashed window
 - o Caught some enforcer hits
 - o Initial window size was wrong
 - o Syntax error in a pattern led to an endless loop
-

- V 3.20
- o First released

1.32 AProf: Question and Answer

- o What is an 'clean' symbol table?

Clean symbol tables include only function entry addresses. Neither intermediate labels are defined nor data symbols in code hunks.

1.33 AProf: Preferences dialog

```
APROF: PREFERENCES DIALOG                                @ { " Exit help " close }
-----
```

Symbol pattern:

Provide a regular expression for symbols you want to exclude from profiling. For a description of Amiga regular expressions see your dos manual.

Patterns can be used to remove compiler generated intermediate labels or functions you aren't interested in. One advantage of removing functions from the symbol table is faster profiling.

Example: 'L#[0-9]' removes symbols starting with an 'L' followed by any number of digits (L1 L00 L4711).

Separate/Combined: (Mode)

Specifies how function timing values are accumulated. If combined is selected, the time value for a function includes the times of all functions that are called by this function. Separate excludes the times of called functions.

Note that if your code includes recursive procedure you must use separate mode. If combined is used, the time for the recursive procedure is wrong.

See @ { "example" link gad-inex }.

Millisecond/Percentual time units: (Units)

Selects the units used for for timing values. If percentual is active, times shown are fractions of overall execution time in percent.

Safe profiling:

If safe profiling is on (selected), then while profiling, every breakpoint must pass additional tests to ensure that it belongs to a function entry or exit (See @ { "dirty symbol tables" link frageantwort }). If off, this tests are not done, resulting in faster profiling. Since it is hard to notice the time spent in doing the tests, it

is recommended to activate safe profiling.

Rexx Unmangler:

Specify the filename of the Rexx procedure to be used for name unmangling.

Some programming languages (eg. C++) offer 'type-safe linking', which is often implemented by coding the argument types into the symbol names (this process is called symbol mangling). Since for humans this symbol names are then hard to read, it is possible to specify a Rexx procedure for translating the symbol back in a more readable form.

See @{"example Rexx unmangler" link rexxample }.

Sort order:

Specify the preferred sort order.

1.34 AProf: Rexx example

```

/* Dummy C Unmangler
 *
 * This demonstrates the basics for AProf unmanglers written in Rexx
 *
 * Don't use this in real life, since this is included in AProf
 * as standard unmangler (No mangler selected in preferences)
 *
 * This unmangler removes a leading '_', if one exists
 */

/* Get the symbol name from AProf and put it in 'symnam' */
parse arg symnam

/* Check if there is a leading '_' */
if "_" = left( symnam, 1 ) then
  /* Remove first char */
  symnam = right( symnam, length( symnam ) -1 )

/* Return result */
exit symnam

```

1.35 AProf: Exec dialog

```

APROF: EXEC DIALOG                                @{" Exit help " close }
-----

```

Command line:

A command line for the profilee can be specified. The command name must not be given.

Example: You have written a program named 'foo', which needs one

argument 'bar' to be specified on the command line. The following steps are needed to get an execution profile of 'foo':

- o Start AProf: aprof foo
- o Select 'Data...' and insert 'bar' as command line
- o Select 'Use'
- o Select 'Start'

Stack size:

Provide a stack size for the profilee.
