

SnoopLibs

COLLABORATORS

	<i>TITLE :</i> SnoopLibs		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		February 6, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	SnoopLibs	1
1.1	Copyright	1
1.2	Contents	1
1.3	Example	3
1.4	Gadgets: Task List.	4
1.5	Gadgets: Function List.	5
1.6	Gadgets: Patch List.	5
1.7	Gadgets: Output.	6
1.8	Gadgets: Patch.	7
1.9	Menu: Load Library	7
1.10	Menu: Output File	8
1.11	Tooltypes	9
1.12	How to add libraries...	9
1.13	Fd2Ls	10
1.14	Argument FormatString	11
1.15	Argument FormatString Example	12
1.16	Warnings	13

Chapter 1

SnoopLibs

1.1 Copyright

Copyright.

SnoopLibs.doc, (c) ASWare 1993, by Ekke Verheul and Dirk Reisig.

REGISTER:

Send US\$ 40.- to receive the ASWare-PD works. The disks will include: Replex, Undelete, Spots, Magnum24, Lynx, MCAnim, Blanks, MandAnim, RXShow, WBPAT, Delta24, Parking, Faster!, AmigaMind, ChangeMode, ScreenModeReq, SnoopLibs, and all the programs we most certainly will be making in the future... including most of the C and assembler sources!

Send your support to:

```

/-----\
|         |
|  ASWare |
| Postbus 2521 |
| 3000 CM Rotterdam |
| the Netherlands |
|         |
\-----/
```

Contents.

1.2 Contents

SnoopLibs.doc, (c) ASWare 1993, by Ekke Verheul and Dirk Reisig. ↔

SnoopLibs was made for developers. It is a tool to find out more about the behaviour of a running task. It does so by making it possible to patch `_any_` function(s) from `_all_` public libraries and print out the calls from that task. The output format and type of information to print are editable by the user.

Contents.

----- How to use SnoopLibs:

Example.

Gadgets: Task List.

Gadgets: Function List.

Gadgets: Patch List.

Gadgets: Output.

Gadgets: Patch.

Menu: Load Library.

Menu: Output File.

Tooltypes.

----- How to add a library.

Intro

The "fd2sl" tool.

Argument formats.

Example.

----- Warnings.

Warnings.

----- How to receive the source.

Register.

1.3 Example

Example:

1. Load the "graphics.sl" file and select the functions "OpenFont", "SetFont" and "CloseFont". Those three names will disappear from the middle (Functions) list and appear in the right (Patch) list. To ease the finding-process you can use the 'o', 's' and 'c' keys.
2. Select "Workbench" in the Task-list as the task you want to watch.
3. Set the patch "ON".
This will open a window with the following text:

```
*** "CloseFont" patched.
*** "OpenFont" patched.
*** "SetFont" patched.
```

4. Now open a device or directory icon to activate the Workbenchtask, and read:

```
OpenFont (TextAttr: 'topaz.font', 8)
SetFont (rp: 0x28C0CC, textFont: 'topaz.font', 8)
CloseFont (textFont: 'topaz.font', 8)
OpenFont (TextAttr: 'topaz.font', 8)
SetFont (rp: 0x28C0CC, textFont: 'topaz.font', 8)
CloseFont (textFont: 'topaz.font', 8)
OpenFont (TextAttr: 'topaz.font', 8)
SetFont (rp: 0x28C0CC, textFont: 'topaz.font', 8)
CloseFont (textFont: 'topaz.font', 8)
OpenFont (TextAttr: 'topaz.font', 8)
SetFont (rp: 0x28C0CC, textFont: 'topaz.font', 8)
CloseFont (textFont: 'topaz.font', 8)
... and so on ...
```

5. Then set the patch to "OFF" and read:

```
*** "CloseFont" patch released.
*** "OpenFont" patch released.
*** "SetFont" patch released.
```

6. To close the window you can change the "Output". This would release the patches also, but they were released before.

What does this example tell us?

- a) The Workbench uses an `OpenFont()`, `SetFont()` and `CloseFont()` call for `_every_` icon it has to display. This may well be one of the reasons our workbench is a bit slow :-)
- b) SnoopLibs can be used to show the name and height of a font when a function receives a pointer to a `Font` or `FontAttr` structure. In fact, SnoopLibs supports redirection arguments for all calls.

Gadgets: Task List.

Contents.

1.4 Gadgets: Task List.

Gadgets: Task list.

If SnoopLibs is running, there will be a list of Tasks in the list called "Task" (how appropriate). Selection of a Task will show the Taskname under the list, and immediately change the function of SnoopLibs. The tested task may be changed even when the Patch is ON!

"Update" the tasklist if you just added a task (or started a program) that you want to snoop. The list is not (yet) updated automatically.

At this moment (version 0.9) it is not possible to select a task that is not yet active, or to select a group of tasks or all. That means that "SnoopLibs" is no substitute for "SnoopDOS" (© Eddy Carroll). This may change in the future, depending on the needs of registered users! (hint, hint :-)

Gadgets: Function List.

Gadgets: Patch List.

Gadgets: Output.

Gadgets: Patch.

Menu: Load Library.

Menu: Output File.

Tooltypes.

Contents.

1.5 Gadgets: Function List.

Gadgets: Function list.

This list will contain all functions of the loaded ".sl" file. You can choose one or more function(s) from this list and load another list until your set of functions-to-patch is complete. The list will be sorted (A-Z) and by using the keys (a-z) you can jump to the right character. This makes it easier to find a function in a long list.

Gadgets: Patch List.

Gadgets: Output.

Gadgets: Patch.

Menu: Load Library.

Menu: Output File.

Tooltypes.

Contents.

1.6 Gadgets: Patch List.

Gadgets: Patch list.

This list holds the functions-to-patch. You can add a function by clicking on that function in the Functions-list, and remove a function by clicking on the function in the Patch-list. Simple. If you remove a function from the Patch-list it will appear again in the Functions-list.

Gadgets: Output.

Gadgets: Patch.

Menu: Load Library.

Menu: Output File.

Tooltypes.

Contents.

1.7 Gadgets: Output.

Gadgets: Output.

Choose "WINDOW" (default), "FILE", or "PRINTER". The patch will go "OFF" if the output is changed, and the output-file (or window) will close.

The output-file (or window) will open when the patch is turned ON.

Gadgets: Patch.

Menu: Load Library.

Menu: Output File.

Tooltypes.

Contents.

1.8 Gadgets: Patch.

Gadgets: Patch.

Switch between "ON" or "OFF". Releasing a patch is always a bit risky, so SnoopLibs takes one second to turn it OFF. Read

Warnings
about this. If a function was patched again after SnoopLibs patched it, SnoopLibs will not release the patch. First release the other patch in that case.

Menu: Load Library.

Menu: Output File.

Tooltypes.

Contents.

1.9 Menu: Load Library

Menu: Load Library.

This option will open the asl filerequester. The default directory will be "slibs", for that's where the ".sl" textfiles are.

If you choose a library from the list, "SnoopLibs" will forget about all functions of the last loaded library, except for those that were selected (t.i. functions in the 'Patch' list). The 'Patch' list can contain any number of functions from up to ten different libraries.

This version (0.9) of Snooplibs comes with six ".sl" files; they contain information about the following six libraries:

dos.library
exec.library
gadtools.library
graphics.library
icon.library

intuition.library

All functions in those six libraries (including V39) can be patched.

Anyone can add libraries to this list, or change the output of the different calls. You can create special (mixed) ".sl" files to ease selection, ect. Read the

How to add libraries...

part

of this manual and the "mix.sl" example in the "slibs" directory.

Menu: Output File.

Tooltypes.

Contents.

1.10 Menu: Output File

Menu: Output File.

This option will open the asl filerequester. The default directory will be "T:" and the default output-file-name is "SnoopLibs.txt".

This file will contain all the text created during a session from the moment you choose "FILE" as the output instead of "WINDOW". If you switch the patch from "ON" to "OFF", the file will remain opened, and can not be accessed from other tasks. The file will close when you choose another output ("WINDOW" or "PRINTER") or when you quit "SnoopLibs".

The "WINDOW" option always opens a console-window like this: "CON:20/20/600/180/ SnoopLibs Output". If you don't like this, open a "FILE" with no directory-string and a "CON" name-string.

Tooltypes.

Contents.

1.11 Tooltypes

The Tooltypes.

Only three tooltypes are supported in this version:

LIBRARY="slibs/name.sl" : The default funtionlist.

MAXPRI=5 : Some tasks have a very high (or low) priority,

MINPRI=-5 : ..and those are risky to patch! Read

Warnings

.

How to add a library.

Contents.

1.12 How to add libraries...

How to add libraries.

When you are a bit familiar with SnoopLibs, and want to patch functions from libraries other than the six in 'slibs', then it's time to read this. But first load one of the ".sl" files to see what they look like... into your editor, yes.

The first line of a snoopl原因-file is always:

```
* SnoopLib: name.library
```

.. where "name.library" holds the name of the discribed library. This line may come back later, with another name, but it must be found on the first line too!

Another "star" line may be:

```
* VERSION 37
```

.. and indicates that the following functions need version 37 (or later) of this library. Any version-number will do. Those functions will not be in the function-list if the library is too old. Use this option!

All information about one function is found on one line. Those lines look like this:

```
FF3A OpenScreen «%s (newScreen: 0x%06lx)» «a0»
```

```

^           ^           ^-----^           ^
Offset  Name           « Formatstring »       « Register(s) »

```

A list with the offset, functionname, init-formatstring and init-registers will be created for you by the

FD2SL

program. This

tool and it's source are (should be) included with SnoopLibs.

After the conversion of the ".fd" file to the ".sl" file there

are some things left to be done by hand. That is hard work, but

I don't see any other way. Those are the things that must be done:

1) Place the " * VERSION #" lines above the functions that need that version of the library. Not all ".fd" file are consequent about this sort of information.

2) Edit all FormatStrings and register-arguments by hand. This is important! FD2SL does some work for you, but not all of it!

Read the

Formats

part.

Contents.

1.13 Fd2Ls

Fd2Sl

Fd2Sl is a special tool to convert a ".fd" file (that comes with a library) to a SnoopLibs ".sl" file. The result (the ".sl" file) is a normal editable ASCII text file.

Fd2sl can only be used from CLI or Shell. The format is:

```
fd2sl [>slibs/libname.sl] <libname.fd> [R]
```

output : The result is printed to stdout.

input : The first argument must be the name of a ".fd" file, containing information about the library to add.

[R] : "Remarks" in a ".fd" file will by default be printed to the ".sl" file. This makes it easier to add the "VERSION" lines. Adding the 'R' flag will remove the remarks.

The result can immediately be used by SnoopLibs or may be improved by hand. Read the

Formats

part.

Contents.

1.14 Argument FormatString

Argument FormatString.

Let's check this out:

```
FFBE SetFont «%s (rp: 0x%06lx, textFont: '%s', %ld)» «a1,10(a0),20(a0).w»
^         ^         ^         ^
Offset  Name   FormatString                               Arguments
```

FormatString: The FormatString is placed between a '«' and a '»'
 ----- ("Alt-(" and "Alt-)") and should start with a '%s'
 because the first argument is the function's name. All other '%'
 signs must have a corresponding register in the arguments-part.
 For the conversion "RawDoFmt()" is used, so here's a part of the
 "RawDoFmt" manual:

FormatString - a "C"-language-like NULL terminated format string,
 with the following supported % options:

%[flags][width.limit][length]type

flags - only one allowed. '-' specifies left justification.
 width - field width. If the first character is a '0', the
 field will be padded with leading 0's.
 . - must follow the field width, if specified
 limit - maximum number of characters to output from a string.
 (only valid for %s).
 length - size of input data defaults to WORD for types d, x,
 and c, 'l' changes this to long (32-bit).
 type - supported types are:
 b - BSTR, data is 32-bit BPTR to byte count followed
 by a byte string, or NULL terminated byte string.
 A NULL BPTR is treated as an empty string.
 d - decimal
 u - unsigned decimal
 x - hexadecimal
 s - string, a 32-bit pointer to a NULL terminated
 byte string. A NULL pointer is treated as an
 empty string.
 c - character.

REMEMBER: All types other than pointers (b and s) must have
 the 32bit-length specified.

Always add the 'l' : "%ld", "%lu", "%lx" and "%lc" are OK.

Arguments: The arguments are placed between a '«' and a '»',
 ----- and reflect the contents of the registers at the
 time the function was called. These values can be used in some
 different ways:

```
1) Direct:            «rp: 0x%06lx» «a1»
2) Indirect:          «NextWindow: 0x%06lx» «(a0)»
3) Indirect with Offset: «textfont: '%s'» «10(a0)»
4) Direct WORD        «XPos: %ld» «d0.w»
5) Indirect WORD     «FontHeight: %ld» «20(a0).w»
6) Direct BYTE        «Flags: 0x%02lx» «d0.b»
7) Indirect BYTE     «Flags: 0x%02lx» «20(d0).b»
```

It is ok to use «20(d0).l» if you prefer to be clear about the argument size, but LONG arguments are the default. Even WORD or BYTE arguments have a "%l" in the FormatString, the real argument-size is always 32-bits! Oh, asm-programmers, don't be surprised about the «20(d0)» notation; even data-registers may be used as pointers here.

Be carefull not to use anything like: «3(a0)» on a M68000 !!!
 Odd offsets can only be used with BYTE values!

Example.

Contents.

1.15 Argument FormatString Example

Argument FormatString Example.

 Let's take the graphics-function OpenFont() and try to make a
 usefull FormatString for it.

OpenFont.

The docs tell us this function takes a pointer to a TextAttr
 structure in register a0:

```
struct TextFont *OpenFont(struct TextAttr *);
                           a0
```

That structure looks like this:

```
struct TextAttr {
    STRPTR ta_Name;       // 0       /* name of the font */
```

```

    UWORD   ta_YSize;    // 4      /* height of the font */
    UBYTE   ta_Style;    // 6      /* intrinsic font style */
    UBYTE   ta_Flags;    // 7      /* font preferences and flags */
};
// 8

```

I've added the offsets of the variables. Those values can be found by using the a0 register like this:

```
« (a0), 4(a0).w, 6(a0).b, 7(a0).b »
```

and the FormatString would look like this:

```
«%s (FontName: '%s', Height %ld, Style: 0x%02lx, Flags: 0x%02lx)»
```

The Style and Flags of a font are not very interesting, so they could be omitted. Whatever you like.

Contents.

1.16 Warnings

Warnings.

- 1) Releasing a patch is always risky. Suppose a task is busy with the patched code when the patch is removed and the code in memory changes ... oops! SnoopLibs tries to overcome that problem by waiting one full second after removing the patches and before returning the memory. That works fine for all tasks with a normal taskpriority. Tasks with a very low priority could become a problem. That's why the "MINPRI" Tooltype was added. Use it with care.
 - 2) Testing the behaviour of a task with a very high taskpriority will result in unpredictable effects. Patching the input-device f.i. leaves you with a weird functioning mouse... The reason for this is ofcourse that patching a function takes some time... some tasks just don't have that time. That's why the "MAXPRI" Tooltype was added. Use it with care.
 - 3) Be aware of "loops". F.i. Patching a device and writing the output to the same device may create a never-ending loop. Releasing the patch will end the outputstream after some time, but only if the priority of that task is not too high.
 - 4) Be very carefull when creating your own ".sl" files. Wrong arguments or redirections (like odd WORD or LONG offsets) may cause the system to crash.
- Did you find a bug? Please let me know about it!

Contents.
