

# The ATRSHMLOG

What's in there ? We are Version 1.1.0 now....

## Table of Contents

Introduction.....	14
The usual chapter you won't need to read.....	14
History first.....	17
How it began, how it was revived, how it goes on.....	17
The next version 1.1.0.....	18
Whom do we have to thank for this ?.....	20
The guys behind this module.....	20
The guys who have made it better.....	20
The basics of the ATRSoft GmbH Shared Memory logging module.....	21
This is the first must read chapter.....	21
Why to build it.....	22
The ways to get a working module.....	22
Definition : BASEDIR.....	22
Definition : Module.....	23
Definition : Support program.....	23
Definition : Library ( THE ).....	25
Definition : Headers.....	25
Definition : Build programs.....	25
Definition : Helper.....	29
Definition : Layer.....	29
Definition : JNI Layer.....	30
Definition : The perl layer.....	31
Definition : The python layer.....	32
Definition : The SWIG layer.....	32
Definition : The area.....	32
Definition : The event.....	33
Definition : The environment.....	33
Definition : The flag files.....	34
How to build it.....	35
The hopefully working way.....	35
After the download.....	35
Check for completeness.....	38
Check for the platform.....	40
First time: build.....	42
First test : create the buffer.....	44
First test : init the area.....	45
First test: run the simplest test program.....	46
First test: getting the log into files.....	47
First time : converting binary to human readable.....	50

When it does not work for you.....	52
Check the C compiler.....	55
Check the C++ compiler.....	57
Check the OS if the things fail to run.....	58
When it does not fit for your needs.....	59
Changes with the build in initialization stuff.....	59
Changes in the code.....	61
Adding stuff.....	62
Now that I have it – how do I use it ?.....	63
The way to implement a simple logging program.....	63
Include stdio.h.....	64
Starting point and parameters.....	65
Calling the library function printf.....	65
Returning from main.....	65
The final end (or not ?).....	66
Adding the module.....	66
Attaching to an area.....	68
Adding the logging.....	70
Compile and test.....	71
The deep stuff.....	73
Measuring a printf.....	78
The big example for the C community.....	80
main.....	81
eval.....	83
exec.....	84
expr.....	84
funcs.....	85
The rest.....	85
The java language support.....	86
How it works.....	86
How to use it.....	87
How to build it in the first place.....	87
The java directory.....	88
The bin directory.....	89
Copy headers and lib from the C module.....	92
Change into your vendor and jdk directory.....	93
Setting the environment.....	98
Building with create_jni_lib.sh.....	99
Testing the jni bridge.....	100
Details.....	101
The python language support.....	108
Python's SEX.....	108
How it works.....	109
How to use it.....	109
How to build it in the first place.....	110
The python directory.....	111
The bin directory.....	112
Copy headers and lib from the C module.....	114
Change into your src directory.....	115
Setting the environment.....	117

Building with create_python_lib.sh.....	118
Testing the python bridge.....	119
Details.....	120
The perl language support.....	124
Perl XS.....	125
How it works.....	125
How to use it.....	126
How to build it in the first place.....	126
The perl directory.....	127
The bin directory.....	128
Copy headers and lib from the C module.....	131
Change into your src directory.....	132
Setting the environment.....	134
Building with create_perl_lib.sh.....	135
Testing the perl bridge.....	136
Details.....	137
The one fits all SWIG approach.....	142
How it works.....	142
How to use it.....	142
How to build it in the first place.....	143
The SWIG directory.....	144
The bin directory.....	145
Copy headers and lib from the C module.....	148
Change into your src directory.....	149
Setting the environment.....	151
Building with create_swig_lib.sh.....	152
Testing the tcl bridge.....	153
Details.....	154
Another platform : CentOS.....	159
Get the compiler to work that you need.....	159
Changing the build scripts.....	161
Testing.....	164
Another platform : FreeBSD.....	165
Another platform : OpenBSD.....	167
Another platform : NetBSD.....	169
Another platform : Solaris.....	170
Another platform : Ubuntu.....	171
Another platform : OpenSUSE.....	172
Another platform : Debian 8.6.....	173
Another platform : SLES.....	174
Another platform : cygwin.....	176
After the unpack.....	178
Prepare headers – NO MORE .....	179
First compile.....	180
The cygserver start.....	182
Create of a buffer.....	184
Making the area with init.....	185
First test with atrshmlogtest00.....	186
Reader for transfer.....	187
Conversion of the binary to human readable form.....	189

Another platform : mingw.....	191
Copy the headers – NO MORE .....	192
Compile with makeall.sh.....	194
End of compile.....	195
Path handling for vanilla cmd.....	196
Creating the buffer.....	197
The init of the area.....	199
First test with atrshmlogtest00.....	200
Starting the reader for memory fetching.....	201
Conversion of the binary into human readable text.....	203
The jni layer for mingw.....	205
Testing the jni bridge for mingw.....	207
What are those numbers for ? Adjustment process ?.....	209
Now we need to know how fast it is.....	209
Now we know how fast it is – can we make it faster ?.....	213
A low throughput scenario.....	214
A scenario for a long term running low throughput program.....	215
A scenario with low throughput and multiple threads.....	215
A scenario with high throughput and small number of threads.....	216
A scenario with very high throughput.....	216
Statistics.....	218
When I change it, what then ?.....	220
The Adjustment.....	221
Changes in the first place in atrshmlog.h.....	221
Changes for internals.....	221
Changes for the code.....	222
Local changes for your own system.....	224
Patches.....	225
Wouldn't it be nice to get this, too ?.....	226
The glory details.....	227
Theory of the module.....	228
The way to log in shared memory – or how to circumvent it.....	231
Cindy's classroom.....	234
Back to work.....	238
A word about fences – or should I say memory barriers ?.....	241
Fence 1.....	242
Fence 2.....	242
Fence 3.....	243
Fence 4.....	243
Fence 5.....	243
Fence 6.....	243
Fence 7.....	243
Fence 8.....	243
Fence 9.....	243
Fence 10.....	244
Fence 11.....	244
Fence 12.....	244
Fence 13.....	244
And now for the gallery – the C module way.....	245
The log buffer.....	250

state.....	250
shmsize.....	250
next_append.....	250
next_full.....	251
pid.....	251
tid.....	251
inittime.....	251
inittimetsc_before inittimetsc_after.....	251
Lasttime lasttimetsc_before lasttimetsc_after.....	251
starttransfer.....	252
acquiretime.....	252
id.....	252
chksum.....	252
safeguard.....	252
number_dispatched.....	252
counter_write0 to counter_write2_adaptive_very_fast.....	252
info.....	252
The area.....	253
shmversion.....	253
shmid.....	253
shmsafeguard.....	253
Shma shmf.....	253
shmcount.....	253
ich_habe_fertig.....	253
Readerflag readerpid.....	253
Writerflag writerpid.....	253
logbuffers.....	254
The tbuff struct in client.....	254
next_cleanup.....	254
next_full.....	254
next_append.....	254
safeguardfront.....	254
Pid tid.....	254
acquiretime.....	255
id.....	255
size.....	255
maxsize.....	255
dispose.....	255
dispatched.....	255
number_dispatched to counter_write2_adaptive_very_fast.....	255
b.....	255
The thread locals.....	256
i.....	256
atrshmlog_idnotok.....	256
atrshmlog_targetbuffer_arr.....	256
atrshmlog_targetbuffer_index.....	256
atrshmlog_shm_count.....	256
strategy.....	256
Autoflush.....	256
atrshmlog_thread_pid atrshmlog_thread_tid.....	256

number_dispatched to counter_write2_adaptive_very_fast.....	256
The slave local.....	257
next.....	257
tid.....	257
g.....	257
The externs.....	257
The macros.....	257
Real code.....	258
atrshmlog.c.....	258
atrshmlog_attach.....	258
atrshmlog_init_shm_log.....	260
atrshmlog_cleanup_locks.....	261
atrshmlog_verify.....	261
atrshmlog_create.....	262
atrshmlog_delete.....	262
atrshmlog_get_area.....	262
atrshmlog_get_area_count.....	262
atrshmlog_get_area_version.....	263
atrshmlog_get_area_ich_habe_fertig.....	263
atrshmlog_set_area_ich_habe_fertig.....	263
atrshmlog_transfer_mem_to_shm.....	263
atrshmlog_read_fetch.....	264
atrshmlog_read.....	265
atrshmlog_alloc.....	265
atrshmlog_il_connect_buffers_list.....	266
atrshmlog_acquire_buffer.....	266
atrshmlog_dispatch_buffer.....	266
atrshmlog_free.....	267
atrshmlog_flush.....	267
atrshmlog_write0, atrshmlog_write1, atrshmlog_write2.....	267
atrshmlog_init_thread_local.....	269
atrshmlog_init_in_write.....	269
atrshmlog_stop.....	270
atrshmlog_turn_logging_off.....	270
atrshmlog_reuse_thread_buffers.....	270
atrshmlog_exit_cleanup.....	271
atrshmlog_create_slave.....	271
atrshmlog_f_list_buffer_slave_proc.....	272
atrshmlog_decrement_slave_count.....	272
atrshmlog_remove_slave_via_local.....	273
atrshmlog_get_next_slave_local.....	273
atrshmlog_turn_slave_off.....	273
atrshmlog_get_env.....	273
atrshmlog_get_env_shmid.....	274
atrshmlog_get_env_id_suffix.....	274
atrshmlog_get_env_prefix.....	274
atrshmlog_set_env_prefix.....	274
atrshmlog_buffers_preallocated.....	274
atrshmlog_il_get_raw_buffers.....	275
atrshmlog_get_logging.....	275

atrshmlog_get_realtime.....	276
atrshmlog_get_statistics.....	276
atrshmlog_sleep_nanos.....	276
atrshmlog_set_event_locks_max.....	276
atrshmlog_init_events.....	277
atrshmlog_get_acquire_count.....	277
atrshmlog_get_autoflush_process.....	277
atrshmlog_get_buffer_id.....	277
atrshmlog_get_buffer_max_size.....	277
atrshmlog_get_buffer_size.....	277
atrshmlog_get_clock_id.....	277
atrshmlog_get_checksum.....	277
atrshmlog_get_env_id_suffix.....	277
atrshmlog_get_event.....	278
atrshmlog_get_event_locks_max.....	278
atrshmlog_get_thread_fence_1.....	278
atrshmlog_get_thread_fence_2.....	278
atrshmlog_get_thread_fence_3.....	278
atrshmlog_get_thread_fence_4.....	278
atrshmlog_get_thread_fence_5.....	278
atrshmlog_get_thread_fence_6.....	278
atrshmlog_get_thread_fence_7.....	278
atrshmlog_get_thread_fence_8.....	278
atrshmlog_get_thread_fence_9.....	278
atrshmlog_get_thread_fence_10.....	278
atrshmlog_get_thread_fence_11.....	278
atrshmlog_get_thread_fence_12.....	278
atrshmlog_get_thread_fence_13.....	279
atrshmlog_get_f_list_buffer_slave_count.....	279
atrshmlog_get_init_buffers_in_advance.....	279
atrshmlog_get_inittime.....	279
atrshmlog_get_inittime_tsc_after.....	279
atrshmlog_get_inittime_tsc_before.....	279
atrshmlog_get_minor_version.....	279
atrshmlog_get_patch_version.....	279
atrshmlog_get_prealloc_buffer_count.....	279
atrshmlog_get_shmid.....	279
atrshmlog_get_f_list_buffer_slave_wait.....	279
atrshmlog_get_statistics_max_index.....	279
atrshmlog_get_strategy.....	279
atrshmlog_get_strategy_process.....	279
atrshmlog_get_tid.....	280
atrshmlog_get_thread_local_tid.....	280
atrshmlog_get_version.....	280
atrshmlog_get_wait_for_slaves.....	280
atrshmlog_set_init_buffers_in_advance_off.....	280
atrshmlog_set_init_buffers_in_advance_on.....	280
atrshmlog_set_buffer_size.....	280
atrshmlog_set_autoflush.....	280
atrshmlog_set_autoflush_process.....	280

atrshmlog_set_checksum.....	280
atrshmlog_set_clock_id.....	280
atrshmlog_set_event.....	280
atrshmlog_set_thread_fence_1.....	280
atrshmlog_set_thread_fence_2.....	280
atrshmlog_set_thread_fence_3.....	281
atrshmlog_set_thread_fence_4.....	281
atrshmlog_set_thread_fence_5.....	281
atrshmlog_set_thread_fence_6.....	281
atrshmlog_set_thread_fence_7.....	281
atrshmlog_set_thread_fence_8.....	281
atrshmlog_set_thread_fence_9.....	281
atrshmlog_set_thread_fence_10.....	281
atrshmlog_set_thread_fence_11.....	281
atrshmlog_set_thread_fence_12.....	281
atrshmlog_set_thread_fence_13.....	281
atrshmlog_set_logging_process_off_final.....	281
atrshmlog_set_f_list_buffer_slave_count.....	281
atrshmlog_set_logging_process_off.....	281
atrshmlog_set_logging_process_on.....	282
atrshmlog_set_prealloc_buffer_count.....	282
atrshmlog_set_f_list_buffer_slave_wait.....	282
atrshmlog_set_strategy.....	282
atrshmlog_set_strategy_process.....	282
atrshmlog_set_thread_fence.....	282
atrshmlog_set_wait_for_slaves_off.....	282
atrshmlog_set_wait_for_slaves_on.....	282
atrshmlog_set_f_list_buffer_slave_run_off.....	282
atrshmlog_init_via_env.....	282
atrshmlog_init_via_file.....	282
NON INLINE CODE.....	282
All files with _flag.c.....	285
All files with _buffer.c.....	285
All remaining files with _list.c.....	285
Best behavior.....	285
Appendix.....	287
Error codes.....	291
atrshmlog_error_ok.....	291
atrshmlog_error_error.....	291
atrshmlog_error_error2.....	291
atrshmlog_error_error3.....	292
atrshmlog_error_error4.....	292
atrshmlog_error_error5.....	292
atrshmlog_error_connect_1.....	292
atrshmlog_error_connect_2.....	292
atrshmlog_error_init_thread_local_1.....	292
atrshmlog_error_mem_to_shm_1.....	293
atrshmlog_error_mem_to_shm_2.....	293
atrshmlog_error_mem_to_shm_3.....	293
atrshmlog_error_mem_to_shm_4.....	293

atrshmlog_error_mem_to_shm_5.....	294
atrshmlog_error_mem_to_shm_6.....	294
atrshmlog_error_mem_to_shm_7.....	294
atrshmlog_error_mem_to_shm_8.....	294
atrshmlog_error_attach_1.....	295
atrshmlog_error_attach_2.....	295
atrshmlog_error_attach_3.....	295
atrshmlog_error_attach_4.....	295
atrshmlog_error_attach_5.....	295
atrshmlog_error_attach_6.....	296
atrshmlog_error_attach_7.....	296
atrshmlog_error_init_in_write_1.....	296
atrshmlog_error_write0_1.....	297
atrshmlog_error_write0_2.....	297
atrshmlog_error_write0_3.....	297
atrshmlog_error_write0_4.....	297
atrshmlog_error_write0_5.....	298
atrshmlog_error_write0_6.....	298
atrshmlog_error_write0_7.....	298
atrshmlog_error_write0_8.....	298
atrshmlog_error_write0_9.....	299
atrshmlog_error_write0_10.....	299
atrshmlog_error_write1_1.....	299
atrshmlog_error_write1_2.....	299
atrshmlog_error_write1_3.....	300
atrshmlog_error_write1_4.....	300
atrshmlog_error_write1_5.....	300
atrshmlog_error_write1_6.....	300
atrshmlog_error_write1_7.....	301
atrshmlog_error_write1_8.....	301
atrshmlog_error_write1_9.....	301
atrshmlog_error_write1_10.....	301
atrshmlog_error_write1_11.....	301
atrshmlog_error_write1_12.....	302
atrshmlog_error_write1_13.....	302
atrshmlog_error_write2_1.....	302
atrshmlog_error_write2_2.....	302
atrshmlog_error_write2_3.....	303
atrshmlog_error_write2_4.....	303
atrshmlog_error_write2_5.....	303
atrshmlog_error_write2_6.....	303
atrshmlog_error_write2_7.....	304
atrshmlog_error_write2_8.....	304
atrshmlog_error_write2_9.....	304
atrshmlog_error_write2_10.....	304
atrshmlog_error_write2_11.....	305
atrshmlog_error_write2_12.....	305
atrshmlog_error_write2_13.....	305
atrshmlog_error_area_version_1.....	305
atrshmlog_error_area_count_1.....	306

atrshmlog_error_area_ich_habe_fertig_1.....	306
atrshmlog_error_get_event_1.....	306
atrshmlog_error_get_logging_1.....	307
atrshmlog_error_get_logging_2.....	307
atrshmlog_error_get_logging_3.....	307
atrshmlog_error_get_logging_4.....	307
atrshmlog_error_create_1.....	308
atrshmlog_error_create_2.....	308
atrshmlog_error_create_3.....	308
atrshmlog_error_create_4.....	308
atrshmlog_error_init_shm_1.....	308
atrshmlog_error_init_shm_2.....	309
atrshmlog_error_init_shm_3.....	309
atrshmlog_error_read_1.....	309
atrshmlog_error_read_2.....	310
atrshmlog_error_read_3.....	310
atrshmlog_error_read_4.....	310
atrshmlog_error_read_5.....	310
atrshmlog_error_read_6.....	311
atrshmlog_error_read_fetch_1.....	311
atrshmlog_error_read_fetch_2.....	311
atrshmlog_error_read_fetch_3.....	312
atrshmlog_error_read_fetch_4.....	312
atrshmlog_error_read_fetch_5.....	312
atrshmlog_error_read_fetch_6.....	312
atrshmlog_error_verify_1.....	313
atrshmlog_error_verify_2.....	313
atrshmlog_error_verify_3.....	313
atrshmlog_error_verify_4.....	313
atrshmlog_error_verify_5.....	314
atrshmlog_error_verify_6.....	314
atrshmlog_error_buffer_slave_1.....	314
atrshmlog_error_get_strategy_1.....	314
atrshmlog_error_set_strategy_1.....	315
atrshmlog_error_get_autoflush_1.....	315
atrshmlog_error_set_autoflush_1.....	315
Statistics.....	316
atrshmlog_counter_time_low.....	316
atrshmlog_counter_time_high.....	316
atrshmlog_counter_attach.....	316
atrshmlog_counter_get_raw.....	316
atrshmlog_counter_free.....	317
atrshmlog_counter_alloc.....	317
atrshmlog_counter_dispatch.....	317
atrshmlog_counter_mem_to_shm.....	317
atrshmlog_counter_mem_to_shm_doit.....	318
atrshmlog_counter_mem_to_shm_full.....	318
atrshmlog_counter_create_slave.....	318
atrshmlog_counter_stop.....	318
atrshmlog_counter_write0.....	319

atrshmlog_counter_write0_abort1.....	319
atrshmlog_counter_write0_abort2.....	319
atrshmlog_counter_write0_abort3.....	319
atrshmlog_counter_write0_abort4.....	319
atrshmlog_counter_write0_discard.....	320
atrshmlog_counter_write0_wait.....	320
atrshmlog_counter_write0_adaptive.....	320
atrshmlog_counter_write0_adaptive_fast.....	320
atrshmlog_counter_write0_adaptive_very_fast.....	320
atrshmlog_counter_write_safeguard.....	321
atrshmlog_counter_write_safeguard_shm.....	321
atrshmlog_counter_write1.....	321
atrshmlog_counter_write1_abort1.....	321
atrshmlog_counter_write1_abort2.....	321
atrshmlog_counter_write1_abort3.....	322
atrshmlog_counter_write1_abort4.....	322
atrshmlog_counter_write1_discard.....	322
atrshmlog_counter_write1_wait.....	322
atrshmlog_counter_write1_adaptive.....	323
atrshmlog_counter_write1_adaptive_fast.....	323
atrshmlog_counter_write1_adaptive_very_fast.....	323
atrshmlog_counter_write1_abort5.....	323
atrshmlog_counter_write1_abort6.....	323
atrshmlog_counter_write1_abort7.....	324
atrshmlog_counter_write2.....	324
atrshmlog_counter_write2_abort1.....	324
atrshmlog_counter_write2_abort2.....	324
atrshmlog_counter_write2_abort3.....	325
atrshmlog_counter_write2_abort4.....	325
atrshmlog_counter_write2_discard.....	325
atrshmlog_counter_write2_wait.....	325
atrshmlog_counter_write2_adaptive.....	325
atrshmlog_counter_write2_adaptive_fast.....	326
atrshmlog_counter_write2_adaptive_very_fast.....	326
atrshmlog_counter_write2_abort5.....	326
atrshmlog_counter_write2_abort6.....	326
atrshmlog_counter_write2_abort7.....	327
atrshmlog_counter_set_slave_count.....	327
atrshmlog_counter_set_clock_id.....	327
atrshmlog_counter_slave_off.....	327
atrshmlog_counter_set_event_locks.....	327
atrshmlog_counter_set_buffer_size.....	328
atrshmlog_counter_set_wait_slaves_on.....	328
atrshmlog_counter_set_wait_slaves_off.....	328
atrshmlog_counter_set_slave_wait.....	328
atrshmlog_counter_set_prealloc_count.....	329
atrshmlog_counter_set_thread_fence.....	329
atrshmlog_counter_create.....	329
atrshmlog_counter_create_abort1.....	329
atrshmlog_counter_create_abort2.....	329

atrshmlog_counter_create_abort3.....	330
atrshmlog_counter_create_abort4.....	330
atrshmlog_counter_delete.....	330
atrshmlog_counter_cleanup_locks.....	330
atrshmlog_counter_init_shm.....	330
atrshmlog_counter_read.....	331
atrshmlog_counter_read_doit.....	331
atrshmlog_counter_read_fetch.....	331
atrshmlog_counter_read_fetch_doit.....	331
atrshmlog_counter_verify.....	331
atrshmlog_counter_logging_process_on.....	332
atrshmlog_counter_logging_process_off.....	332
atrshmlog_counter_set_strategy.....	332
atrshmlog_counter_set_strategy_process.....	332
atrshmlog_counter_set_event.....	333
atrshmlog_counter_set_env_prefix.....	333
atrshmlog_counter_exit_cleanup.....	333
atrshmlog_counter_flush.....	333
atrshmlog_counter_logging_process_off_final.....	333
atrshmlog_counter_turn_logging_off.....	334
atrshmlog_counter_init_in_advance_on.....	334
atrshmlog_counter_init_in_advance_off.....	334
atrshmlog_counter_reuse_thread_buffers.....	334
atrshmlog_counter_set_autoflush.....	335
atrshmlog_counter_fence_alarm_1.....	335
atrshmlog_counter_fence_alarm_2.....	335
Thread local statistics.....	335
Strategy.....	336
atrshmlog_strategy_discard.....	336
atrshmlog_strategy_spin_loop.....	336
atrshmlog_strategy_wait.....	336
atrshmlog_strategy_adaptive.....	337
atrshmlog_strategy_adaptive_fast.....	337
atrshmlog_strategy_adaptive_very_fast.....	337
Environment setting.....	338
ATRSHMLOG.....	339
ATRSHMLOG_ID.....	339
ATRSHMLOG_COUNT.....	339
ATRSHMLOG_INIT_IN_ADVANCE.....	340
ATRSHMLOG_STRATEGY.....	340
ATRSHMLOG_STRATEGY_WAIT_TIME.....	340
ATRSHMLOG_DELIMITER_VALUE.....	340
ATRSHMLOG_EVENT_COUNT_MAX.....	341
ATRSHMLOG_BUFFER_SIZE.....	341
ATRSHMLOG_PREALLOC_COUNT.....	341
ATRSHMLOG_SLAVE_WAIT_NANOS.....	342
ATRSHMLOG_SLAVE_COUNT.....	342
ATRSHMLOG_WAIT_FOR_SLAVES_ON.....	342
ATRSHMLOG_CLOCK_ID.....	343
ATRSHMLOG_CHECKSUM.....	343

ATRSHMLOG_FENCE_1 to 13.....	344
ATRSHMLOG_LOGGING_IS_OFF_AT_START.....	344
ATRSHMLOG_EVENT_NULL.....	344
ATRSHMLOG_EVENT_ONOFF.....	345
ATRSHMLOG_FETCH_COUNT.....	345
ATRSHMLOG_WRITE_COUNT.....	346
ATRSHMLOG_ALLOC_ADVANCED.....	346
Functions to use before attach.....	347
atrshmlog_set_env_prefix.....	347
atrshmlog_set_event_locks_max.....	347
atrshmlog_set_buffer_size.....	347
atrshmlog_set_f_list_buffer_slave_count.....	347
atrshmlog_set_clock_id.....	347
atrshmlog_set_wait_for_slaves_on.....	347
atrshmlog_set_wait_for_slaves_off.....	347
atrshmlog_set_f_list_buffer_slave_wait.....	348
atrshmlog_set_prealloc_buffer_count.....	348
atrshmlog_set_strategy_process.....	348
atrshmlog_set_thread_fence_1 to atrshmlog_set_thread_fence_13.....	348
atrshmlog_set_init_buffers_in_advance_on.....	348
atrshmlog_set_init_buffers_in_advance_off.....	348

# Introduction

As always...

## The usual chapter you won't need to read

As I had the luxury to read a book about the korn shell I stumbled over a very unusual intro. The guy addressed the need for such a thing to be – not there. Simply put: no one reads today such a first chapter without info.

So what ? I still think I have to start somehow, so this is my try.

Everyone who has made development in the software business had at some point to do some thing that we loosely call logging.

To be a bit more specific, there was in the early days of computing a time when a program had to run without interaction with a user. So the programs who crashed or did some minor damage were a real pain in the ass.

No way to get information about what went wrong – so the guys started to do the write to a special place thing. Write at this address in memory a 1 or 2 and when the thing crashed check for the values in the places to see what was done and what not ....

OK. Then the first problem oriented languages arrived at the scene, and the guys who wrote programs changed. Before you could have easily mistaken them for doctors or people of the scientific stuff – wearing white was in at those days. Perhaps it was really some kind of BLACK science to make a program run. Or a library. Or an include, or or or ....

But from then on the guys thought no longer in terms of memory addresses and bpl and js codes, but more in equations and something that was a similar thing for nearly all languages, the assignments. And of course string handling.

And so the way to find out what the problem was when the program didn't do what it should was to write some information on a thing called the console.

This was a special kind of file in fact. You could code your program, let it compile, then run and after this you got not only the output for your program, but also the output of the console while the thing was running.

This was a big step into a new direction – you could now wear normal clothing, no more white dress....

And then it came to the idea to make that console thing even some kind of working in two directions. The input for the program was no longer a thing of using files, but now also a thing to make so called input statements in the program that could take information from the console and deliver it to the program when it needed it.

In fact this was simply a two file approach, with one getting the output and the other holding the input, but it was nearly as if you could now talk to the program.

Wow. Interaction by using files .... but it worked, and then when the time came to try the real thing – making the OS of the system giving real key strokes from real users to a program, that in return gave output to a thing that was readable for the user without wearing a white cloth – a new understanding of the use of the console began.

It was the time when the guys changed again. This time they were some highly misunderstood geeks who looked Star Trek and discussed in their free time about making game programs and not about the newest car of the president....

It was time for the so called real time operating system with multi user capability.

So the console became the place where god – or root – could talk to his child and get its answers.

Sometimes the so called super user could give his new thoughts to the thing, sometimes give it a boost for processing time, and sometimes even set it to sleep .

All with the use of the console, which was now the preferred way to interact with a program.

OK. Now the guys from the processor guild came in, and when they saw how this console thing worked they instantly implemented in their processors special instructions, so that god – or root – could faster speak to the program. The CPU of this time had special IO ops for CIN and COUT .

And this could have been a nice and prospering finish, if it weren't for the guys from Xerox which invented the GUI and the stuff that today stands between nearly every user and his program – the Desktop or touch thing metaphor to handle things without having a console....

Back to reality.

Today we have 99 % of the users of computers in a shape that is far from everything that was needed to use a computer in the first 50 years of their existence. So the metaphor of the desktop and the today touch screen analogon is clearly the way these want to communicate with their programs.

The console is still living in the areas of program development and operating background systems like databases or today's application servers.

So we use today the console mostly in the area of IT pros who have the need for a direct communication without any good looking GUI ballast.

If you are in this business, you know why its a good thing to have a console – mostly for inspection of the things the program in question does, sometimes to interact while it is running. Sometimes to even kill it. And most the time to automate things with a script language then.

Today you can get access to a console on every so called UNIX derivat, even the Android and the Mac OS X systems, and if you insist for the so called fenster;plural systems too.

So for the pro the console is a vital thing, and for the rest of the users something they don't need at all.

Back to logging.

For most programs its a fact that they produce a lot of output on a console. So this thing is simply not the right one to hold that much information. Its better to use a log file.

Files for logging were near the same time introduced thing with the console.

While the console is for interaction and lets say starters in business, the log file is for the long term the better thing.

OK, its ugly to keep all there files in place for inspections, and even worse its not clear if you get what you want – or need – but its the only way to do it else. So the log file is the one thing every system developer and operator has to know. And has to use.

Logging is in this sense the approach of the developer to give useful information for himself or for other developers at a well defined place. And later the operators who know near all or nothing about the system can try to get information from them too.

Logging is in this sense a time shifted one way communication from the developer to someone else who has to figure out what happened in the program.

We usually use a file for this, and logging into a file is today as easy as making three lines of code and then do what you think is needed to be done for the content to log.

So logging is even so important that the latest champion in the ring for application development got its own well known file based logging module log4j.

Logging is the standard way to get info from your database if anything is not right, and from your application servers, even from the OS itself.

This said, there is a small but very persistent class of problems that the file based logging cannot solve. One is the logging of high speed applications, and one is the logging of crash systems.

This is the area where the approach of logging into a memory region comes again in place. It was the first way to check for a broken program – writing at defined memory locations some codes – and is in high performance environments the only thing that can be used to come up with the log without slowing down the process to an absurd slow speed of file logging.

So there we are.

We need a log without a file in one of these special areas, and one is the area of programs that make them self heavy use of files on a low level, like the shell programs does.

If you really want to log a fast system you will recognize that the logging itself can be a problem bigger than the internal of the program itself – especially in multi threaded programs.

So if you need a log and you cannot use the files approach – here is an alternative.

If you need a crash log – well, that's different. Then you will be better suited with one of the many shared memory buffer loggers from the internet community.

# **History first**

This is for the first version.

Version 1.0.0

## **How it began, how it was revived, how it goes on**

OK. The first time i had the idea was somewhere in 1991 when i made on an HP UX 7 system several programs to simply check what was able to do with that new UNIX thing – at least for me, who had only seen simple monitor based OS's and one super high class supercomputer with its own front end computers that filled a whole floor at the university of cologne in the 1980's.

I had some trouble to get it right for a lisp interpreter – the thing crashed in more or less silly ways and i could not find it right on the top with the debugger that I had – so I wrote some data to a buffer in shared memory and when my program was long gone could check for its content with a dumper.

Then I finished the educational stuff, and came into a company that made use of shared memory for a complex cache system – it was horrible to see the guy who tried to do it right, and so i made myself the plan to develop something better than the first thing to help – but it didn't came to that. Only some raw sketches survived.

In 1997 it was time to catch up with a team that made some very interesting developments on the newest IBM hardware driven by the AIX system. There they had the need for high speed logging – and it was a multi thread system, too – and accepted my idea to make the logging not on file base, but with a shared memory buffer and a dumper program.

It worked well. At least for this project.

After switching to new duties in new companies I didn't need the log – there were plenty solutions in those projects, and I think at least three time the crash dump thing, and so I simply used the stuff that was given for the logging from some senior programmers.

It was in early 2016 when the logging thing came up again. A project needed some help with performance problems and stuff like debugging scripts in korn shell environments.

Logging from a shell is easy – use a file and echo to it. But logging without changing the scripts, and worse not changing the file handler semantics is best non trivial.

So I gave the fact credit that the shell code was somehow open source and tried to make a logger module for the thing.

Basic stuff was done in a week, and after doing some rough changes I got a thing that was an internal log of the shell operations on a level where no set -x ever could reach ...

It was in fact a disaster when it came to a simple loop running a counter from one to a million. At

best one fifth of the speed. That was BAD.

So I started to make some additional effort for the logging – use of multiple buffers, use of alternate buffer writing and in the last time some trials with buffered logging.

After seven months – the project had decided that it could live with the performance and never needed an internal log – the thing was gone to a very different place. I now had alternate buffers used for threads, supporting threads inside to do the housekeeping and a multi threaded queuing reader that did make use of atomics all over the place. So much for comparison of this thing with a crash dumper or a simple ring buffer write for a program.

It is now nearly end of 2016, and I have added at least basic support for the java user and the python and perl and SWIG thing right after the thing is documented and made it to GitHub – perhaps someone is really interested and can use it ?

## The next version 1.1.0

Now it is the 1.1.0. Have made some ajustments in the header after I got it run on nearly a dozen systems. And yes, even my friend Dr. C.H. made it on the fenster;plural.

So we have now a simpler way to set the build env – only the rigth dot file needed.

For the module itself only minor changes. The usual things when you found something has to be made better....

Interested ?

The autoflush thing. Set it on and you can make it directly in the area – or at least to the slave . That's nice for the post mortem analyze of bad things.

The checksum to help to find fence problems - or better when you need a fence.

The slave functions to handle the slave list and its entries.

The flush that only dispatches the non empty buffers.

The reuse and turn off things.

That alone would be a new release, but most interesting is now the new test thing in tests. And we have a driver in t\_test.sh now in place.

So you can start to test before you put it in your stuff. OK, I have already some additional tests on my list for 1.2.0, but for now the new tests are a first step in the right direction.

So after some changes here and there and rough 8 new functions also the documentation needed an update.

And of course the layers. Still fighting the test game with jni, perl and python. Only swig will be off topic here – its in perl done and depends on the language, so simply use the perl as an inspiration – or sent me your thing and I integrate it like perl.

For the layers most things are related to the C module – as always. But for jni I made also a new write – using startindex and length is convenient when you have to log only last part of a big string.

Together with the big bunch of binaries in unsupported it has been a bigger step.

For 1.2.0 most things will be fixing attach logic problems, better synchronization in the lists and the environment. And a SQL and a NO SQL logger.

But for 1.1.0 we are here now.

# **Whom do we have to thank for this ?**

## **The guys behind this module**

Well, that's me in the first place. And of course the guys at DB Systel AG at Frankfurt am Main who had the need for a korn shell programmer in the first place, namely Stefan Künstler and Thorsten Beilke.

And there is the guy who always thinks in java terms when he discusses things with me, Dr. Christoph Höffner, who made the java thing a first target after the korn shell didn't make it.

## **The guys who have made it better**

Well, that's for now only a placeholder. Give me a post card if you have any ideas in that way. Perhaps you will see your name here in a later version.

# **The basics of the ATRSoft GmbH Shared Memory logging module**

## **This is the first must read chapter**

The module is implemented in C. To be precise its ISO 9899:2011, that's C 11. The supporting programs are implemented in C and C++ - some of them can be of C99 and C++11.

A C++ port is on my todo list. Comes later.

Use of any stuff in the module is free, its given to you with the perhaps least restrictive of all open source licenses, the APACHE license. Please understand that this is done to make it possible for you to use it in any environment you need to. It is not done to make you rich. So you can in fact sell this thing for money, but you cannot change the fact that you are not the creator and so the thing is owned by the company who sponsored most of its development.

This is the ATRSoft GmbH.

This will be the last time you see this name in the documentation. The letters ATR come from the initials of the programmer who did it, and its a mere coincidence that the letters match to the first three of the name of the company.

This is the legal section, so you can be sure now that you can use the module, even in a compiled form in your production code, sell it together with it and don't have to plan for a big lawyer stuff in the future. The only thing you cannot do is claiming you made it – that's all.

The module consists of a source code module in C – so its possible to make changes if you need them. You have the control of the changes, and if they are good I will take over them in the code base – if you want it and it makes sense to me.

## Why to build it

To make a long story short: I had bad experience with binary stuff on Linux over time. In the beginning you have it easy – you build and check on some other flavors and that's it.

Over time things start to change. After some years even best crafted software was broken for the build. Binaries simply throw bad references at the start up – or worse, didn't run after that or worst did produce wrong results.

So a simple and robust approach for the build and source code was the answer. And yes, when its source code its open source for me, too.

So no tree of binary artifacts that can be used out of the box – only some in the unsupported tree.

Instead a simple build with some scripts and that's all. Even the much loved fenster;plural system can make it with the help of cygwin and mingw today to a real scripting paradise.

So building from source it is.

## The ways to get a working module

We have to start with some definitions here, so that you can understand the rest of the documentation.

### Definition : BASEDIR

This is the directory you will unpack the tar ball or zip or download from GitHub. There is no single root directory as you normally get, so its up to you to decide if you need it. I use an account as base, so I haven't done the root directory. I don't need it. But its a bit messy there indeed.

If you are in a multi user system and you can create an account then simply do it. I have mine named in short shmlog, so that should do it in most cases.

If you cannot make an account switch to a directory you can make the unpack thing and create an own root directory.

In the documentation I will call this the BASEDIR.

If it is an account or a separate directory does not matter, but for keeping things simple you should not use it for other things. The scripts are simple, and if you mix other things in it could end in a mess, so please sent me no reports of failed builds when you do this with other systems code inside the BASEDIR. I will ignore that.

If its needed I will refer to the thing simply as BASEDIR, meaning you can replace it with your actual path to the thing and i will simply stay with that.

You will need rough 50 MB for the thing, and this is a big reserve for the build and a simple test. When you plan to do a speed test its more in GB – the log of an atrshmlogtest03 can easy end up in

GB sizes....

## **Definition : Module**

The module consist of the compiled code from the C source code files. The files holding its code are atrshmlog.c and all sources in the impls subdirectory of the BASEDIR. After you have unpacked the tarball or downloaded it you can find the files in src and there in impls.

We use the term module most of the time to refer to the compiled code, the C functions and data structures.

There is sometimes a second C file in place, for example for java the jni file atrshmlogjnipackage.c . This in NOT the module, even if it seems to act like it from the view of the user, in this case the java developer.

## **Definition : Support program**

There is a bunch of support programs. At least when you use the plain C code version, you have not only the module, but also those programs in place. They are having the source code in the files with the program name ending in the .c suffix. So its easy for you to talk about the source code and meanwhile to actually think of the compiled program.

The exceptions of these are the scripts that use perl and shell. They are support programs in this sense. But there is no c source code. Its the script itself that is executed.

Here is a list and a short abstract.

- atrshmlogcalc  
The script to analyze converted output files of the log.
- atrshmlogcheckcomplete  
The script to check the completeness of the source code distro together with the files.txt list.
- atrshmlogchecksystem  
The script to analyze the kind of platform, architecture, flavor, compiler and cross-compile you have in place.
- atrshmlogconv  
The script to convert the binary log files from the reader into a human readable text file.  
Works for a whole directory tree.
- atrshmlogconvert  
The program that converts one binary log file into a human readable text file and optionally create the thread statistics file too.
- atrshmlogcreate  
The program that creates a shared memory buffer for use as a logging area.

- **atrshmlogdefect**  
The test tool to show offsets in the area for variables and buffers and for making byte changes for testing.
- **atrshmlogdelete**  
The program to delete a shared memory buffer, so the system can reuse the memory and the given ID for the buffer.
- **atrshmlogdump**  
The program to read out the area and write it into a binary. IT only reads, it does not change.
- **atrshmlogerror**  
The script to convert a list of error numbers into more readable information.
- **atrshmlogfinish**  
The program to clean up resources of the area so that you no longer use them and can delete the area – or reuse it.
- **atrshmloginit**  
The program to initialize a shared memory buffer with the resources that are needed to use it as an area.
- **atrshmlogoff**  
The program to switch off the system wide logging via the flag in the area.
- **atrshmlogon**  
The program to switch on the system wide logging via the flag in the area.
- **atrshmlogreaderd**  
The program to transfer log from the area to the file system.
- **atrshmlogreset**  
The program to reinitialize the area if needed.
- **atrshmlogsignalreader**  
The program to set an information for a reader in the area.
- **atrshmlogsignalwriter**  
The program to set an information for a writer in the area.
- **atrshmlogsort**  
The script to combine log files into one text files on base of the folder that they are in and ordered by the start time for the content in the resulting file – which has the fixed name prot.txt.
- **atrshmlogstat**  
The script to get a rough estimation about the time profile of your logging.

- atrshmlogstopreader  
The script to stop the atrshmlogreaderd via signal.
- atrshmlogverify  
The program to check for the correctness of the area.

## **Definition : Library ( THE )**

The module is compiled and then a library is build. Its the library that holds the module binary code. Its name is libatrshmlog.a and its a static link library. The footprint is rough 300 k , so I think you can live with a static link. If you need a dynamic one check the script buddylib.sh and change it.

Binary versions will be no part of the distro for now. If you think you cannot live with this, contact me and I will see what I can do for you. And don't forget to check the unsupported first....

The library is used in the linking of some programs, so it is also a test for using the thing.

You should use the library with your own code, but its also possible to link in the objects – just what you need.

## **Definition : Headers**

The module consists of the C source code. To work it needs not only the library, but also the two headers atrshmlog.h and atrshmlog\_internal.h. The headers have been there for a long time, because I felt uneasy with doing all the structure and define stuff in an C file or put it in the interface header. So the internal header was there to hold that stuff. For the interface as a client the atrshmlog.h is all you need. See the demo programs atrshmlogtest00 and atrshmlogtest01 for this.

If you really need more – you try to implement an own converter, or an own reader – you can use the atrshmlog\_internal.h header to get the info that you will need.

The most important thing you have to do before you compile the module is to set up the correct defines in the headers, so check the adjustment chapter for this. And don't forget the platform include – without this you cannot use the headers.

## **Definition : Build programs**

The build of the module and the layers is mostly done with simple script programs. This has the advantage that things are easy and you can simply adapt the module and the layers to your target platform.

And there is no problem with any dependency stuff that could create inconsistent builds. We do it simply from ground on up complete.

The disadvantage is that you can not use the classical approach – configure, make, make install.

There is simply no make file.

No make ?

Yes, no make.

The module has to be build complete – so a make is something you don't need in using the module, only in development if you need the advantage of modularization of the build process. And you need dependency's with a make – which is something that could be confused in a bad environment.

And yes, I had some rough time to get hands on the buildenv for the korn shell, and yes I had two times bad luck in the past for programs I need and the make was simply no longer working.

So no make.

But for the development it showed that the compile cycle was so short that a dependency steering was not needed. Simply build from ground did the job in less than 30 seconds (on a Linux box, cygwin and mingw is a different story...).

So the build was made for the module with a central steering script.

Its the makeall.sh.

For the stuff that is at least suspected to be specific I created some helper scripts. So the compile of the C source files is done with one script, the linking of the programs with another.

Later on the C++ programs came in and so I introduced a script for this too.

And we have one initial step with the execution of a variable setter script, the dot.platform.sh.

This sets some basic info's for the rest of the scripts.

For a supported platform a version with the name of the platform is delivered in the BASEDIR too.

But for now we have only those scripts in the build

- makeall.sh
  - The script that builds the module from ground up.
- g99.sh
  - The script gets the C source name and delivers a compiled object code with the basename and uses the C 11 compiler of your choice. Parts can be done in C 99 – which was the first approach – but only C 11 can do all stuff.
- g++14w.sh
  - The script gets the name of the C++ source and compiles it into a program, so the module is linked as the library to it.
- ell.sh
  - The script gets the name of a program or its object file and links the object file with the module object to the program.
- buddylib.sh

The script builds the library after the compiler has made the object code files.

- `buddydoc.sh`

The script starts the documentation creation process. For now it simply starts doxygen.

There are some files that are used from the `makeall.sh` to decide what to do, they are simple files that contain only one name in a line and can be seen as the steering mechanism that `makeall.sh` does its job.

The files are

- `shmbinfiles`
- `shmbininternalfiles`
- `shmcfiles`
- `shmCPPfiles`
- `shmtestcfiles`

That's a very strange system, but it does the job.

If you have to change it, simply leave the original in a copy tree, and make the changes for the files in place.

The `makeall.sh` uses the bash shell as its interpreter, you can use a different shell by changing the replacement codes for the new shell.

I tested the ksh version Version AJM 93u+ 2012-08-01 by simply changing the shell bang and it worked. So if you don't have the bash try the ksh instead for it.

For the `g99.sh`, `g++14w.sh` and `ell.sh` the change to ksh works the same.

So if you cannot use a bash you can use the ksh instead. This should do the job for most UNIX's that do not have a bash installed.

If you have only a shell – well, you get it. Change the replacement stuff to the old glory basename replaces and you are in again.

For the fenster;plural system we use a cygwin in this document, so the bash is on board here.

If you really need to leave the path of the so called posix systems you can try to re-implement the scripts with the thing you have on the platform. This is normally some thing similar to a shell that execute script code. Of course you can use the multi platform script languages of your choice too.

In the later versions I hope to support perl and python.

If you really insist you can also use the command .com / cmd.exe things. Simply switch to the syntax of the thing and do the makeall.cmd with it. Its not so much complicated, but i will not try this for now. Simply don't have a working fenster;plural box with the compiler of it – and as long the compiler does not support C11 I won't even try it.

Mac OS X should do the job with the on board shell – its a bash as far as I know for now.

For the rest: use your posix conform gnu upgraded system and make a run for the makeall.sh, take the code from g99.sh, ell.sh and g++14w.sh and duplicate it and you are there.

Additional scripts are there too.

- Attrshmlogcheckcomplete

This is a perl script, you should need only the executable perl itself for it. It checks together with the base info in files.txt for the existence of the files of the distro. You should run this as the very first script after you have unpacked a distro.

- Attrshmlogchecksystem

This is a perl script, same as for the check complete. This script tries to figure out what your platform is. This is done best after you have unpack the distro and tested for its completeness.

- Cleanall.sh

This is a little helper to get rid of generated stuff if you need to start from a fresh ground. Its used best after you encounter a problem in the build and need to restart form step 1.

- seal.sh

This script makes the files in the tree read only. So you have a relative safety that your work is not overwritten accidentally after you have done development in the tree.

- Unseal.sh

That's the thing you need to make all things writable in a tree, the opposite of the seal.sh.

- Packdistro.sh

If there is a need to pack the things together that constitutes the distro you can use this. In case you have a nasty problem and no way to solve – pack the distro and then sent it to me for analyze. At least a source code inspection can be done this way.

- The layer scripts

Every layer has its own additional build scripts. They are done in the layer chapters. So for now I only mention that there are script too.

- t\_test.sh

Last but not least our newbie – making tests with one command line .

## **Definition : Helper**

There are some helpers in the scripts. They are here to do things that you could do by a line or two of code on the shell, but that would be more work than we want to do. Its most for the conversion of files and the sort, but also the platform check and the completeness check and the calc are helpers in this sense.

For the code I use what I think is best. So if you insist you can make a program instead. Or use a different script language. Its up to you. But in the module I will try to use only the helpers for the things I have defined above – bash and ksh for scripts, perl for the more complicated stuff.

## **Definition : Layer**

Beside the use of the module in a C environment – which can be also the C++ environment as I have it in my case on the x86\_64 Linux systems with gnu compilers – there is a set of so called layers .

They try to make the module usable for other language environments.

The most prominent is the java layer, which is based on the jni part of java. So I call it the jni layer. There are more layers in this version, we have perl and python and for the SWIG to make these happy that can make use of SWIG – its as far as I know a best supported C to anything else bridge software. So you find there a tcl for example.

A layer consists at least of a thing that the target environment uses as the interface. In case of jni its the java class that defines also the native methods. In case of the python it will be the python module itself. For the rest I don't know for now.

Sometimes there is also an additional artifact needed. For the jni we need not only the java class, but also the byte code and an intermediate layer implemented in C and – after compiler resulting in - a library. So this is the jni library, don't mix it with the module library. You can check for the names, I will try at least to use different names for the different layers – jni in the jni, py in the python, pl in the perl and so on. So it should be possible to have all libraries in place and still not end up in a mixed mess for linking and loading.

For other layers simply sent me a post card and I will see what I can do.

## **Definition : JNI Layer**

For the support of the java community I have created a jni layer that reside in the BASEDIR sub directory java. You have the basic parts in the sub directory src.

The layer is far from perfect, but it gives you at least access to the functions of the module. Be sure you understand it before you try to chance things.

To make things difficult there are several java implementations, so I took the last one I had to deal with – oracle jdk 1.8.0\_66 – and gave it a start.

So you will find for my implementation in the java directory a sub directory oracle, there a jdk1.8.0\_66.

If I support different jdk's I will put the implementation aside in a vendor and version specific directory, the rest will be hopefully the same.

For now I support

- oracle jdk 1.8.0 66 for fedora 23
- oracle jdk 1.8.0 102 for cygwin mingw
- fedora openjdk-1.8.0.111 for fedora 23
- IBM java-x86\_64-80 for fedora 23

You find a bin and a doc and a src directory in there.

For the bin you find some scripts that help to implement the layer.

The scripts are there for reference, not for use.

In the doc folder you find some stuff that helps you at least to take the first step in making a jni work. Its a simple tutorial, so you need to check for the documentation of the jdk too. But its a start if you are new in the business to make jni, so don't ignore it if you are a newbie – like me.

In the src directory you find the needed stuff to build actually the jni layer.

The directory contains of the usual tree for the package the jni layer is located, the two classes that are supported in this version, and the directory to hold the includes on C source side and the directory to hold the resulting library with the jni bridge and at last the directory with the includes.

So the files are

- dot.java.sh

The environment variables are set in the first step when you want to do something by this file. So if you need a different environment setting change it. It mainly sets the JAVA\_HOME variable. For the PATH things are clear. There is also the name of the target system, its the name of the include directory in the jni include directory.

- `create_jni_lib.sh`  
The script that does all the needed things in one run. You need to have the environment up – best use the `.java.sh` for it – and the rest is then done step by step by the script. You have to execute this after you have copied the library and headers in place.
- `atrshmlogjnipackage.c`  
The jni bridge code. This is mostly the hand crafted thing you develop after you got the header from the `javah` tool. Be sure you know what you do if you change it.
- `compile_jni_stub.sh`  
The script to make the jni bridge together with the module into a loadable library. Its simple and needs the new library as the first argument. The rest are the C source and library you need.
- `compile_to_class_package_version.sh`  
The script that compiles the java class into the byte code file. The byte code is then used on java side, while the bridge code is used from the native stubs to call the C functions of the module. Making this simple is wrong, so check your jdk documentation if you need to do something different.
- `create_header_package_version.sh`  
The script that created the jni bridge header file from the byte code of the class. Be sure you know your jdk, then change it if needed.
- `start_package_log.sh`  
The script starts a simple test run. So you see how you have to handle the paths – there are binary load, library load and java paths – and does the simple call to the test class main method.
- `wrong_create_header_package_version.sh`  
Consider this a warning. You can call the tool in a different way, but the resulting header is wrong – at least for me. Compare to the correct one. And then check for your jdk what you need.

After this stuff you have the `includes` directory, that holds the module C source headers and gets the jni bridge header from the create header thing.

Last is the package tree with the java code file for the class `ATRSHMLOG` and `ATRSHMLOGTest` ( how surprising ...) which are holding the Java class for the module access and the helper test class to do some basic tests. An a subpackage tests with the pendants to the C module `t_test` programs.

Details to the layers are in the chapters for the layers.

## Definition : The perl layer

The parts you need to make the perl interpreter using the module. In this case its the SWIG

definition file, the helper wrapper C code, the generated wrap C code, the generated pm file and the shared library.

## **Definition : The python layer**

The parts you need to make the python interpreter using the module. In this case its the module C code file, the helper include for the module file and the shared library.

## **Definition : The SWIG layer**

The parts you need to make the SWIG generating the things for using the module. In this case its the SWIG definition file, the helper wrapper C code, the generated wrap C code, the shared library and any optional file made for the target language.

## **Definition : The area**

The module uses a shared memory buffer. This buffer is given from the OS with a special OS dependent call. So you get a buffer at some address in your address space. This buffer has to be initialized. So the buffer then contains sometimes resources that are bound via handles or structures or simple pointers.

In the development I tried first mutex's, then condition variables, then atomics. All can – but must not – have an OS dependent resource that is bound to something in the OS controlled memory. So I decided not to combine the acquisition of the memory and the initialization. This is a thing that I normally sell as separation of responsibility's to people who ask me why I did it this way.

And so I do this again.

Separate the responsibility's.

One thing is getting the buffer, another thing is the initialization, another the deinitialization – call it delete, finally or whatever – and the last thing the delete of the buffer – which is indeed a simple give back to the OS.

So I need a term for the buffer when it is initialized. This is different from the state when it is only raw memory.

This is then the AREA.

Its simply the initialized and ready to operate buffer.

Keep in mind that it could be on a system that you end dead if you simply give back the buffer without deinitialization.

For now I think that the use of atomics makes it harmless at best. So I skip the finish normally by myself – but your platform could see this different. So check your documentation about the use of

whatever is used in the module if you are not sure. Or simply use the finish program and its functionality.

## **Definition : The event**

This is a bit stolen from the IBM world and the Oracle Database world. Its the meaning of “a place in your target system where you log and its state”.

So an event is an location in the target system – for example the first thing in a C main function after doing the initialization.

And it has a kind of state – actually I use a char of the C system, so in theory at least 256 possible states.

In practice there are only two states. State 0 and state 1.

State 0 is non logging. So an event with state 0 does simply not log.

State 1 is logging. So the event will log.

In this sense its a kind of on off switch for a special log located at an defined point in your target system.

To make this theoretical brabbel a bit more sense simply check the logging macros ATRSHMLOG\_WRITE in the atrshmlog.h header.

## **Definition : The environment**

This means a special set of environment variables. The module uses a prefix and then different suffix's to build the names that are needed. Its possible to set the prefix different, one way via changing the modules define in a header, another using a function call before the initialization of the process side module parts, and another with setting the one and only variable you should not set – its the prefix itself and then its value is used as the new prefix.

So if you check in the module you will see that there is a get for environment variables that have the prefix part in the name first (oha, that's a prefix !) and if you need you can make use things to change the prefix. For the suffix its a define, so you are only having the option to recompile the thing to change that. Of course you are then up to yourself if something not matches in your setup of a use of the module – but this is another story.

The prefix is used in the variable initialization, so if you don't have the environment in place you can use the flag files instead. Then you do the things with files, and its a bit slower, but its the best I could come up in case of a logging login shell that had to use the module. Setting environment variables so that a login shell uses them is at best administrator stuff and could compromise the whole system, so I leave this to you. Use in that case the flag files instead.

## **Definition : The flag files**

Using the flag files is a replacement for the use of environment variables. It was needed to do this to make it possible to use the module for a login shell. I think that's the only place you will need it.

This said – if you have problems with using the environment variables – you can switch at least to this way.

To do so is simple, but a bit boring, too. Simply check for the naming you need for the environment variable, create a file with that name and a .TXT suffix, and give it the right content. Everything is done in numbers, so you have simply write the number in question into the file. A little example would be

```
ATRSHMLOG_ID=123456
```

```
export ATRSHMLOG_ID
```

for the environment way, and

```
echo 123456 > ATRSHMLOG_ID.TXT
```

for the flag file.

The flag files are used when no environment variable is set – but only the ID makes the difference. So you cannot mix them. Its either the env or the flag files else.

# How to build it

From scratch of course....

## The hopefully working way

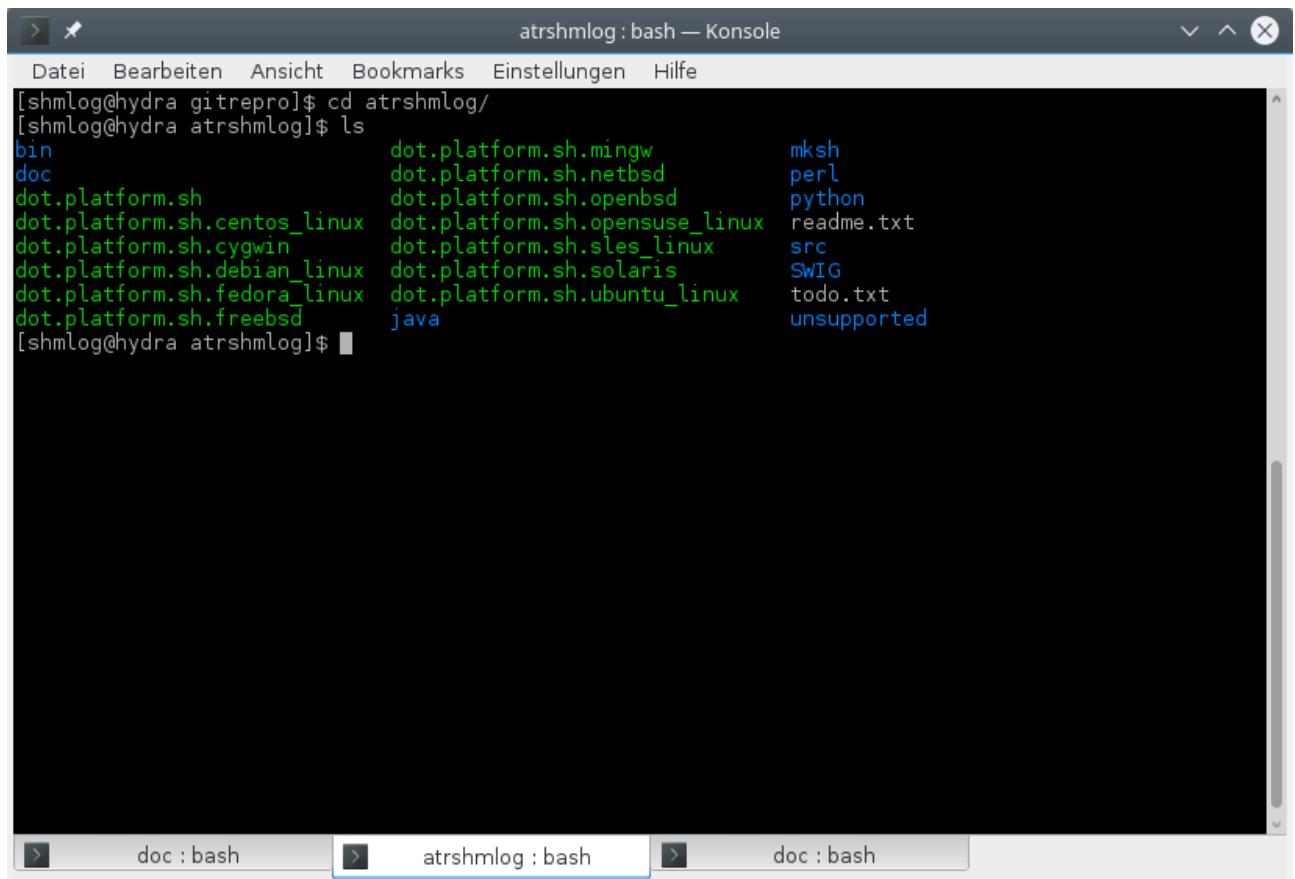
OK. We begin as simple as possible.

## After the download

I assume you got the module as an tar ball or a zip file, or you have made a GitHub download.

So the thing is located at the BASEDIR ( see above if you missed something here ...).

We open our OS shell for the user that has access. It looks in my case like this.



The screenshot shows a terminal window titled "atrshmlog : bash — Konsole". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". The terminal prompt is "[shmlog@hydra gitrepro]\$. The user runs the command "cd atrshmlog" followed by "ls". The output shows a directory structure:

Directory	Content
bin	dot.platform.sh.mingw
doc	dot.platform.sh.netbsd
dot.platform.sh	dot.platform.sh.openbsd
dot.platform.sh.centos_linux	dot.platform.sh.opensuse_linux
dot.platform.sh.cygwin	dot.platform.sh.sles_linux
dot.platform.sh.debian_linux	dot.platform.sh.solaris
dot.platform.sh.fedora_linux	dot.platform.sh.ubuntu_linux
dot.platform.sh.freebsd	java
	mksh
	perl
	python
	readme.txt
	src
	SWIG
	todo.txt
	unsupported

The terminal prompt "[shmlog@hydra atrshmlog]\$" is shown at the bottom. Below the terminal window, there is a taskbar with three tabs: "doc : bash", "atrshmlog : bash" (which is active), and "doc : bash".

Illustration 1: The BASEDIR after unpacking the tar ball....

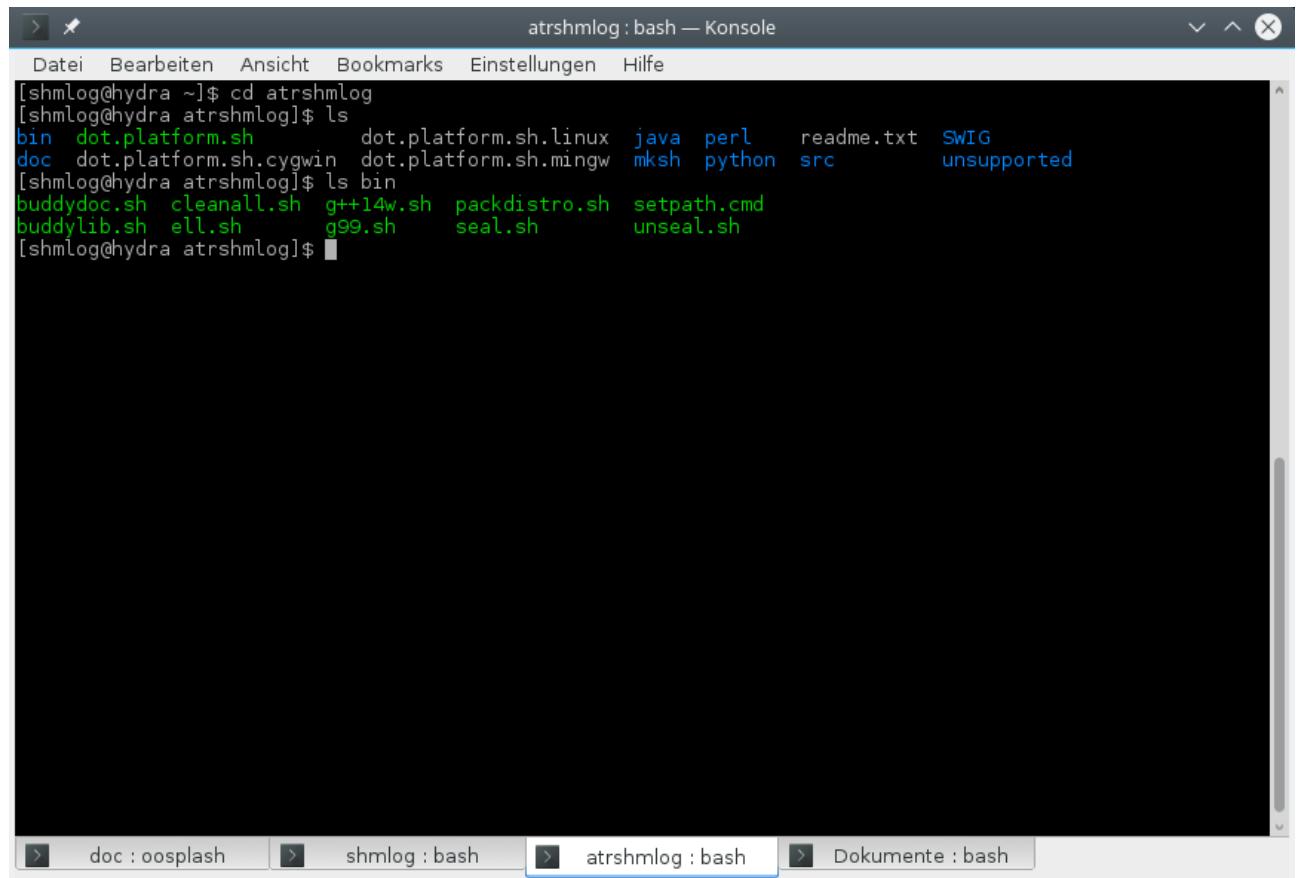
For starters, there are the directories bin, doc, java, mksh, perl, python, src, SWIG.

The unsupported contains stuff that I not officially support, but you can use it.

For the sake of a fast success ignore the rest ....

We check now the bin, it should be in the path of the user that's logged in, so the scripts there should be in place and executable.

Well here is what I get from an ls bin



The screenshot shows a terminal window titled "atrshmlog : bash — Konsole". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". The terminal prompt is "[shmlog@hydra ~]\$". The user runs the command "cd atrshmlog" followed by "ls". The output shows the following files in the bin directory:

```
[shmlog@hydra ~]$ cd atrshmlog
[shmlog@hydra atrshmlog]$ ls
bin  dot.platform.sh      dot.platform.sh.linux  java  perl    readme.txt  SWIG
doc  dot.platform.sh.cygwin  dot.platform.sh.mingw  mksh  python  src        unsupported
[shmlog@hydra atrshmlog]$ ls bin
buddydoc.sh  cleanall.sh  g++14w.sh  packdistro.sh  setpath.cmd
buddylib.sh   ell.sh     g99.sh    seal.sh       unseal.sh
[shmlog@hydra atrshmlog]$
```

Below the terminal window, there is a tab bar with four tabs: "doc : oosplash", "shmlog : bash", "atrshmlog : bash" (which is the active tab), and "Dokumente : bash".

*Illustration 2: The bin directory with the build scripts*

Looks OK for me.

You should have only the regular scripts in place here.

OK. We can now switch to the place where the things have to be used. Its the src directory.

Looks like this

The screenshot shows a terminal window titled "src : bash — Konsole". The window contains a command-line session:

```
[shmlog@hydra gitrepro]$ cd atrshmlog/
[shmlog@hydra atrshmlog]$ ls
bin                      dot.platform.sh.mingw      mksh
doc                      dot.platform.sh.netbsd     perl
dot.platform.sh          dot.platform.sh.openbsd    python
dot.platform.sh.centos_linux dot.platform.sh.opensuse_linux readme.txt
dot.platform.sh.cygwin     dot.platform.sh.sles_linux  src
dot.platform.sh.debian_linux dot.platform.sh.solaris   SWIG
dot.platform.sh.fedora_linux dot.platform.sh.ubuntu_linux todo.txt
dot.platform.sh.freebsd    java                     unsupported
[shmlog@hydra atrshmlog]$ ls bin
buddydoc.sh  cleanall.sh g++14w.sh packdistro.sh setpath.cmd
buddylib.sh  ell.sh    g99.sh    seal.sh    unseal.sh
[shmlog@hydra atrshmlog]$ cd src
[shmlog@hydra src]$ ls
alreadythere           atrshmlogcreate.c    atrshmlogreaderd.c  makeall.sh
ass2.c                 atrshmlogdefect.c   atrshmlogreset.c   shmallbinfiles
ass2.s                 atrshmlogdelete.c   atrshmlogsignalreader.c shmallfiles
ass.c                  atrshmlogdump.c     atrshmlogsignalwriter.c shmbinfiles
ass.s                  atrshmlogerror      atrshmlogsort       shmbininternalfiles
atrshmlog.c            atrshmlogfinish.c   atrshmlogstat       shmcfiles
atrshmlogcalc          atrshmlog.h        atrshmlogstopreader shmcPPfiles
atrshmlogcheckcomplete atrshmloginit.c   atrshmlogverify.c  shmtestcfiles
atrshmlogchecksystem   atrshmlog_internal.c d1
atrshmlogchecksystem.cmd atrshmlog_internal.h Doxyfile      tests
atrshmlogconv          atrshmlogoff.c    files.txt        t_test.sh
atrshmlogconvert.c     atrshmlogon.c    impls
[shmlog@hydra src]$
```

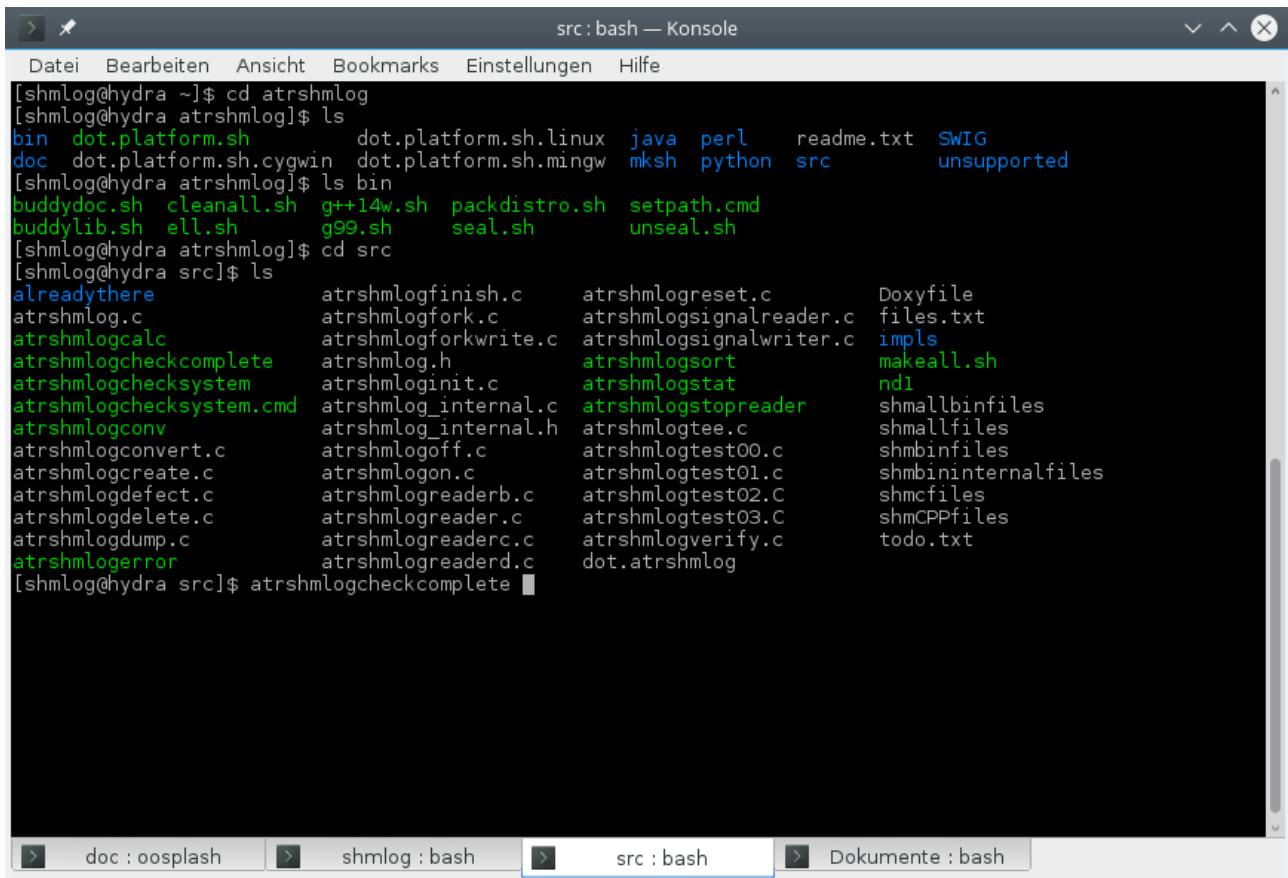
The terminal has three tabs at the bottom: "doc : bash", "src : bash" (which is active), and "doc : bash".

*Illustration 3: The clean src directory*

Well. That's better. Everything in place, only one unsupported helper in here....

## Check for completeness

We check now for a complete distro with the atrshmlogcheckcomplete script.



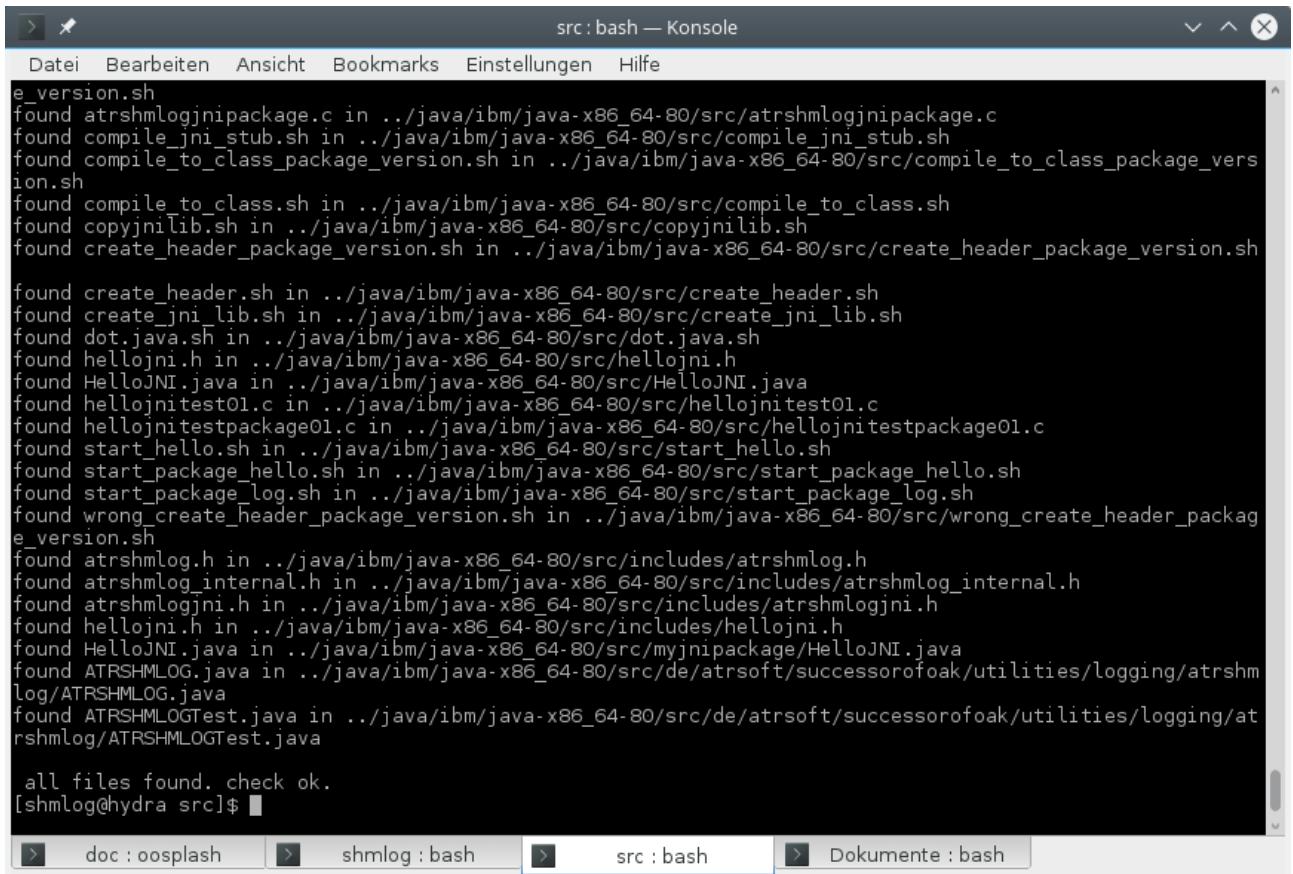
```
src : bash — Konsole
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
[shmlog@hydra ~]$ cd atrshmlog
[shmlog@hydra atrshmlog]$ ls
bin dot.platform.sh          dot.platform.sh.linux  java perl  readme.txt  SWIG
doc dot.platform.sh.cygwin   dot.platform.sh.mingw  mksh python  src      unsupported
[shmlog@hydra atrshmlog]$ ls bin
buddydoc.sh  cleanall.sh  g++14w.sh  packdistro.sh  setpath.cmd
buddylib.sh  ell.sh       g99.sh    seal.sh     unseal.sh
[shmlog@hydra atrshmlog]$ cd src
[shmlog@hydra src]$ ls
alreadythere           atrshmlogfinish.c    atrshmlogreset.c    Doxyfile
atrshmlog.c            atrshmlogfork.c    atrshmlogsignalreader.c files.txt
atrshmlogcalc          atrshmlogforkwrite.c atrshmlogsignalwriter.c impls
atrshmlogcheckcomplete atrshmlog.h        atrshmlogsort        makeall.sh
atrshmlogchecksystem   atrshmloginit.c   atrshmlogstat        ndi
atrshmlogchecksystem.cmd atrshmlog_internal.c atrshmlogstopreader  shmallbinfiles
atrshmlogconv          atrshmlog_internal.h atrshmlogtee.c      shmallfiles
atrshmlogconvert.c     atrshmlogoff.c    atrshmlogtest00.c   shmbinfiles
atrshmlogcreate.c      atrshmlogon.c     atrshmlogtest01.c   shmbininternalfiles
atrshmlogdefect.c      atrshmlogreaderb.c atrshmlogtest02.c   shmcfiles
atrshmlogdelete.c     atrshmlogreader.c  atrshmlogtest03.c   shmCPPfiles
atrshmlogdump.c        atrshmlogreaderc.c atrshmlogverify.c  todo.txt
atrshmlogerror         atrshmlogreaderd.c dot.atrshmlog

[shmlog@hydra src]$ atrshmlogcheckcomplete
```

Illustration 4: Check for complete script start

The start is done as usual. Execute it.

It checks out, everything is in place.



The screenshot shows a terminal window titled "src : bash — Konsole". The window contains the following text output:

```
e_version.sh
found atrshmlogjnipackage.c in ../java(ibm/java-x86_64-80/src/atrshmlogjnipackage.c
found compile_jni_stub.sh in ../java(ibm/java-x86_64-80/src/compile_jni_stub.sh
found compile_to_class_package_version.sh in ../java(ibm/java-x86_64-80/src/compile_to_class_package_version.sh
found compile_to_class.sh in ../java(ibm/java-x86_64-80/src/compile_to_class.sh
found copyjniLib.sh in ../java(ibm/java-x86_64-80/src/copyjniLib.sh
found create_header_package_version.sh in ../java(ibm/java-x86_64-80/src/create_header_package_version.sh

found create_header.sh in ../java(ibm/java-x86_64-80/src/create_header.sh
found create_jni_lib.sh in ../java(ibm/java-x86_64-80/src/create_jni_lib.sh
found dot.java.sh in ../java(ibm/java-x86_64-80/src/dot.java.sh
found hellojni.h in ../java(ibm/java-x86_64-80/src/hellojni.h
found HelloJNI.java in ../java(ibm/java-x86_64-80/src>HelloJNI.java
found hellojnitest01.c in ../java(ibm/java-x86_64-80/src/hellojnitest01.c
found hellojnitestpackage01.c in ../java(ibm/java-x86_64-80/src/hellojnitestpackage01.c
found start_hello.sh in ../java(ibm/java-x86_64-80/src/start_hello.sh
found start_package_hello.sh in ../java(ibm/java-x86_64-80/src/start_package_hello.sh
found start_package_log.sh in ../java(ibm/java-x86_64-80/src/start_package_log.sh
found wrong_create_header_package_version.sh in ../java(ibm/java-x86_64-80/src/wrong_create_header_package_version.sh

found atrshmlog.h in ../java(ibm/java-x86_64-80/src/includes/atrshmlog.h
found atrshmlog_internal.h in ../java(ibm/java-x86_64-80/src/includes/atrshmlog_internal.h
found atrshmlogjni.h in ../java(ibm/java-x86_64-80/src/includes/atrshmlogjni.h
found hellojni.h in ../java(ibm/java-x86_64-80/src/includes/hellojni.h
found HelloJNI.java in ../java(ibm/java-x86_64-80/src/myjni/package/HelloJNI.java
found ATRSHMLOG.java in ../java(ibm/java-x86_64-80/src/de/atrsoft/successorofoak/utilities/logging/atrshmlog/ATRSHMLOG.java
found ATRSHMLOGTest.java in ../java(ibm/java-x86_64-80/src/de/atrsoft/successorofoak/utilities/logging/atrshmlog/ATRSHMLOGTest.java

all files found. check ok.
[shmlog@hydra src]$
```

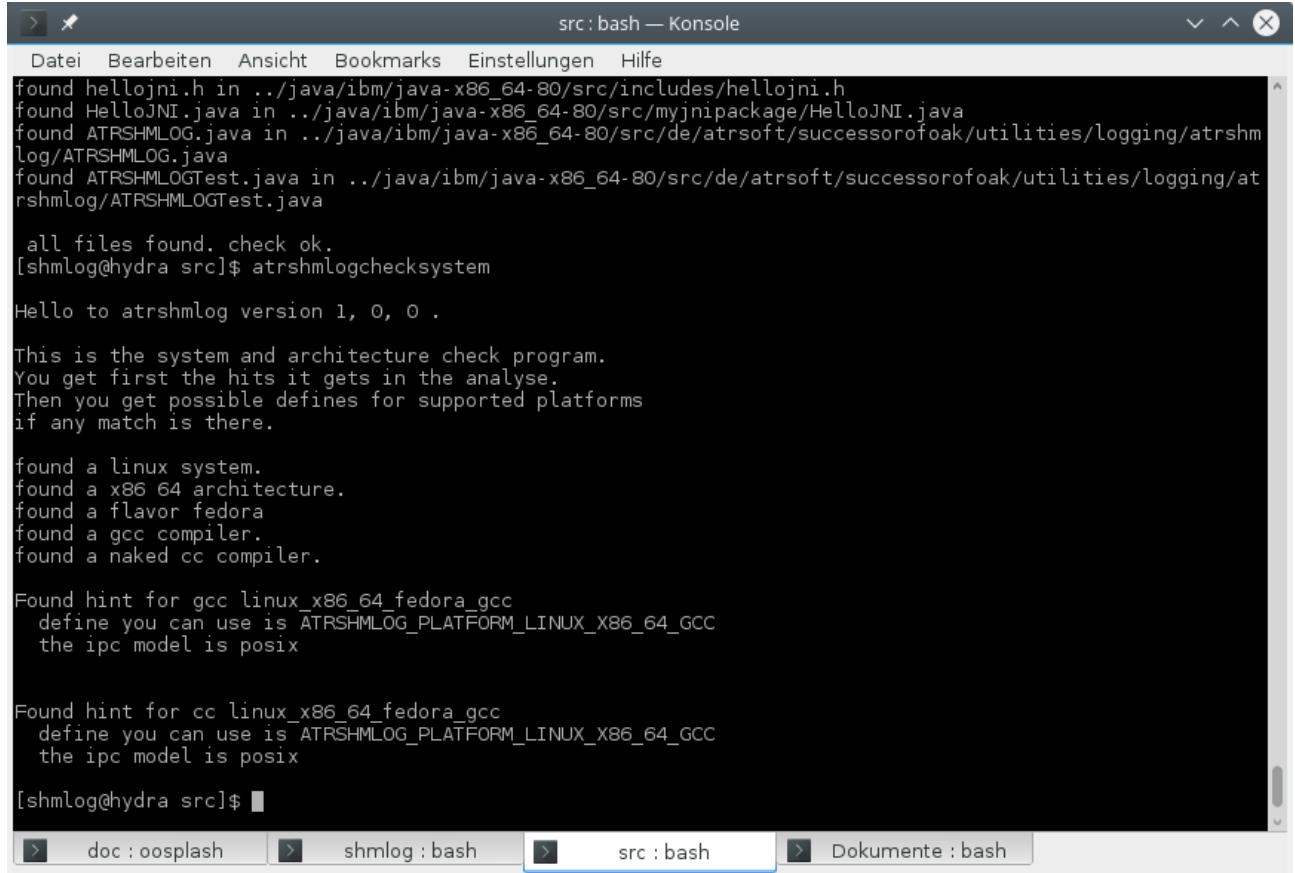
The terminal window has tabs at the bottom: "doc : oosplash", "shmlog : bash", "src : bash" (which is currently selected), and "Dokumente : bash".

*Illustration 5: Result for the check complete script.*

So we can go on with this.

## Check for the platform

OK. We can now check for the platform we use with atrshmlogchecksystem. I spare us the execution, only the outcome here.



```
src : bash — Konsole
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
found hellojni.h in ../java(ibm/java-x86_64-80/src/includes/hellojni.h
found HelloJNI.java in ../java(ibm/java-x86_64-80/src/myjnipackage>HelloJNI.java
found ATRSHMLOG.java in ../java(ibm/java-x86_64-80/src/de/atrsoft/successorofoak/utilities/logging/atrshm
log/ATRSHMLOG.java
found ATRSHMLOGTest.java in ../java(ibm/java-x86_64-80/src/de/atrsoft/successorofoak/utilities/logging/at
rshmlog/ATRSHMLOGTest.java

all files found. check ok.
[shmlog@hydra src]$ atrshmlogchecksystem

Hello to atrshmlog version 1, 0, 0 .

This is the system and architecture check program.
You get first the hits it gets in the analyse.
Then you get possible defines for supported platforms
if any match is there.

found a linux system.
found a x86_64 architecture.
found a flavor fedora
found a gcc compiler.
found a naked cc compiler.

Found hint for gcc linux_x86_64_fedora_gcc
  define you can use is ATRSHMLOG_PLATFORM_LINUX_X86_64_GCC
  the ipc model is posix

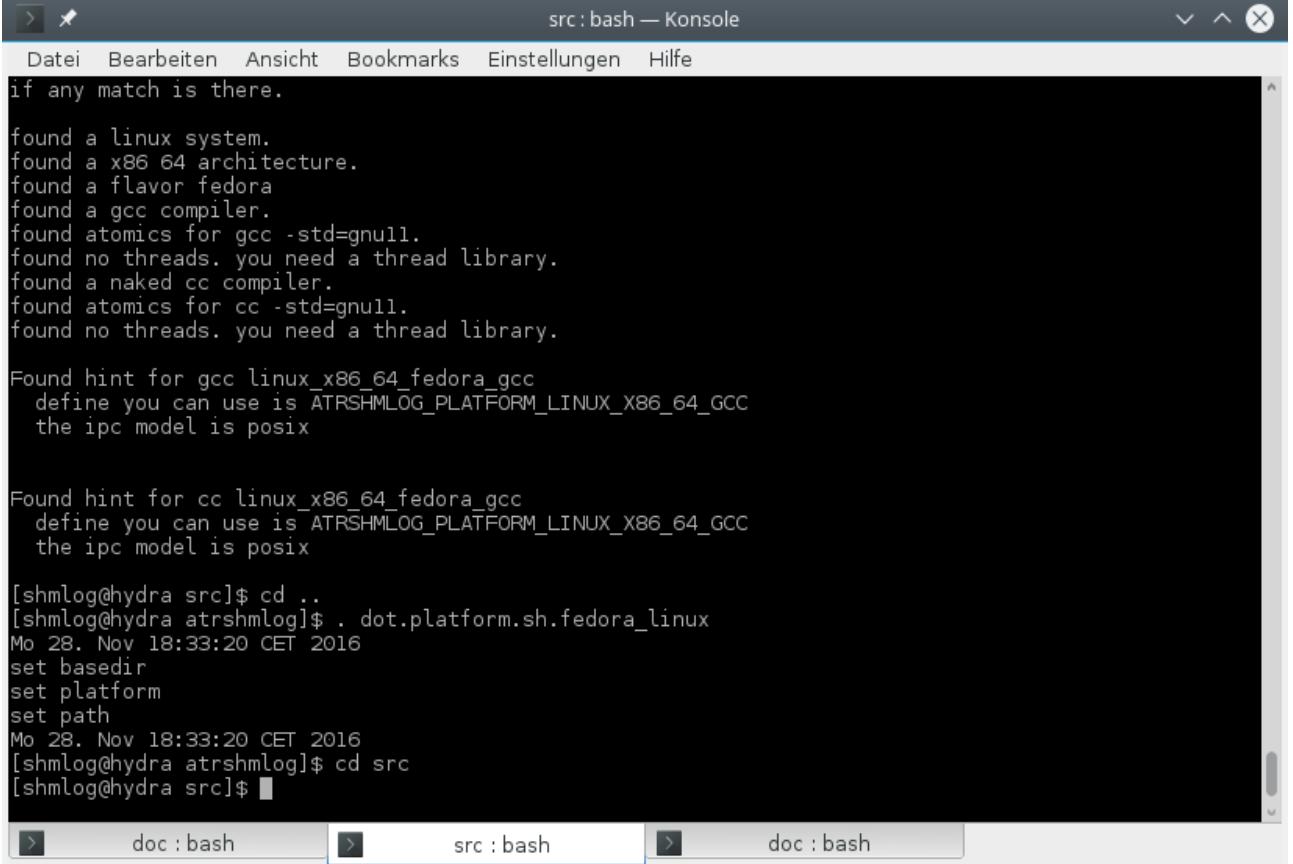
Found hint for cc linux_x86_64_fedora_gcc
  define you can use is ATRSHMLOG_PLATFORM_LINUX_X86_64_GCC
  the ipc model is posix

[shmlog@hydra src]$
```

*Illustration 6: Analyze the system with the check system script*

That's a Linux, a x86\_64, flavor is fedora, and we have a gcc and a cc in place, both can be used as supported platform with the define ATRSHMLOG\_PLATFORM\_LINUX\_X86\_64\_GCC set to 1 in the header. So we can use the existing dot file for that platform, dot.platform.sh.fedora\_linux.

Now we can load the environment variables for the platform. We switch back to BASEDIR. Then we source the already there dot.platform.sh.fedora\_linux file (for my box – yours depend on your system). Then we switch back. Be sure to make the cd because the script has to be used in its directory.



The screenshot shows a terminal window titled "src : bash — Konsole". The menu bar includes "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". The terminal output is as follows:

```
if any match is there.  
found a linux system.  
found a x86_64 architecture.  
found a flavor fedora  
found a gcc compiler.  
found atomics for gcc -std=gnu11.  
found no threads. you need a thread library.  
found a naked cc compiler.  
found atomics for cc -std=gnu11.  
found no threads. you need a thread library.  
  
Found hint for gcc linux_x86_64_fedora_gcc  
define you can use is ATRSHMLOG_PLATFORM_LINUX_X86_64_GCC  
the ipc model is posix  
  
Found hint for cc linux_x86_64_fedora_gcc  
define you can use is ATRSHMLOG_PLATFORM_LINUX_X86_64_GCC  
the ipc model is posix  
  
[shmlog@hydra src]$ cd ..  
[shmlog@hydra atrshmlog]$ . .dot.platform.sh.fedora_linux  
Mo 28. Nov 18:33:20 CET 2016  
set basedir  
set platform  
set path  
Mo 28. Nov 18:33:20 CET 2016  
[shmlog@hydra atrshmlog]$ cd src  
[shmlog@hydra src]$
```

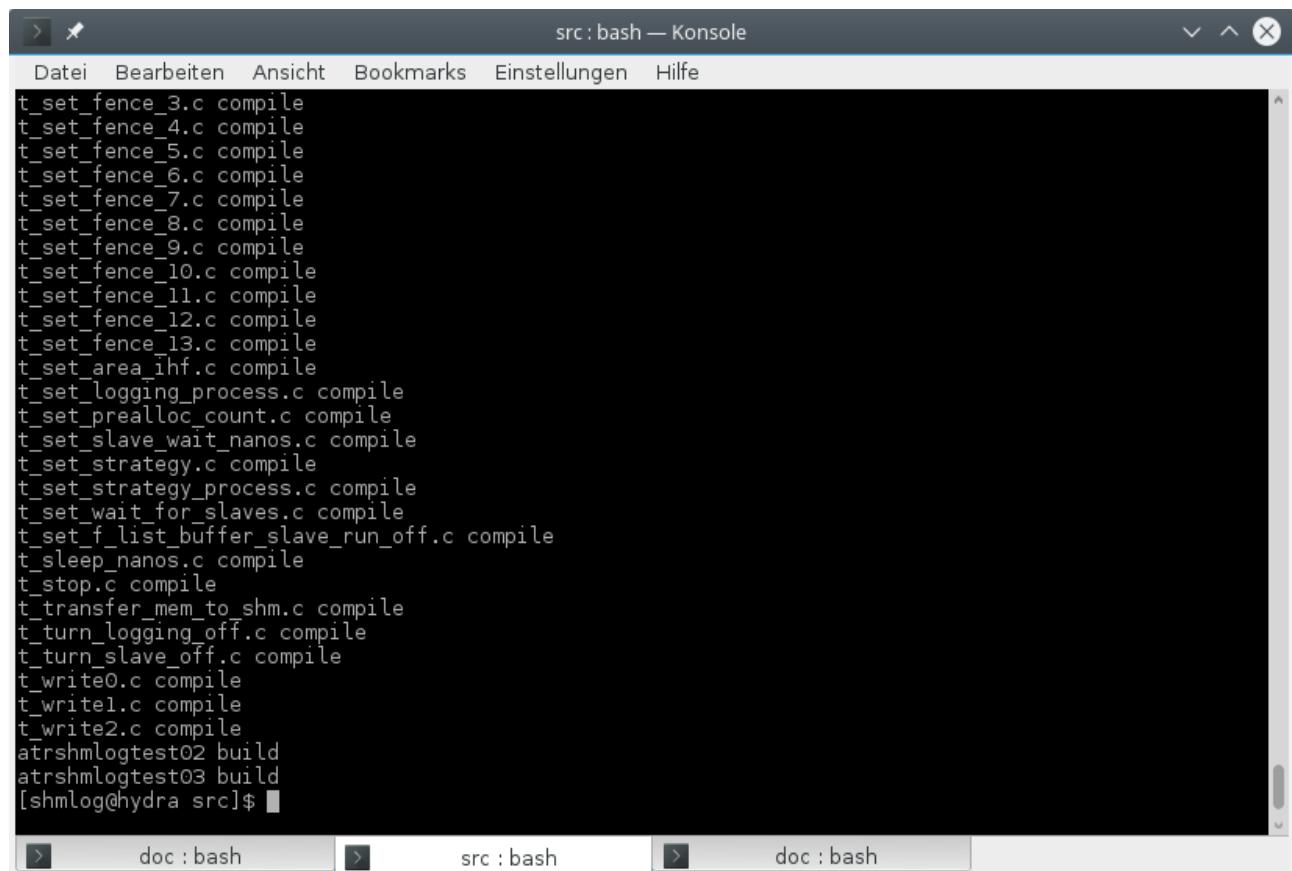
*Illustration 7: Setting the right environment*

So we have now the environment variables in place.

## First time: build

Now we make the module with the one and only call to makeall.sh.

After entering makeall.sh and then return ....



The screenshot shows a terminal window titled "src : bash — Konsole". The window contains a list of compilation commands for various source files. The list includes:

```
t_set_fence_3.c compile
t_set_fence_4.c compile
t_set_fence_5.c compile
t_set_fence_6.c compile
t_set_fence_7.c compile
t_set_fence_8.c compile
t_set_fence_9.c compile
t_set_fence_10.c compile
t_set_fence_11.c compile
t_set_fence_12.c compile
t_set_fence_13.c compile
t_set_area_ihf.c compile
t_set_logging_process.c compile
t_set_prealloc_count.c compile
t_set_slave_wait_nanos.c compile
t_set_strategy.c compile
t_set_strategy_process.c compile
t_set_wait_for_slaves.c compile
t_set_f_list_buffer_slave_run_off.c compile
t_sleep_nanos.c compile
t_stop.c compile
t_transfer_mem_to_shm.c compile
t_turn_logging_off.c compile
t_turn_slave_off.c compile
t_write0.c compile
t_write1.c compile
t_write2.c compile
atrshmlogtest02 build
atrshmlogtest03 build
[shmlog@hydra src]$
```

The terminal window has three tabs at the bottom: "doc : bash", "src : bash" (which is currently active), and "doc : bash".

*Illustration 8: After the makeall.sh finished*

That was not so bad. Everything in place now, the documentation created by doxygen right in place and last but not least the module compiled, the programs there and the library too.

Well, we check it.

```
[shmlog@hydra src]$ ls
t_transfer_mem_to_shm.c compile
t_turn_logging_off.c compile
t_turn_slave_off.c compile
t_write0.c compile
t_write1.c compile
t_write2.c compile
atrshmlogtest02 build
atrshmlogtest03 build
[shmlog@hydra src]$ ls
alreadythere
ass2.c
ass2.s
ass.c
ass.s
atrshmlog.c
atrshmlogcalc
atrshmlogcheckc
atrshmlogcheckcomplete
atrshmlogchecksystem
atrshmlogchecksystem.cmd
atrshmlogconv
atrshmlogconvert
atrshmlogconvert.c
atrshmlogconvert.o
atrshmlogcreate
atrshmlogcreate.c
atrshmlogcreate.o
atrshmlogdefect
atrshmlogdefect.c
[shmlog@hydra src]$ ■
atrshmlogdefect.o      atrshmlogoff.c      atrshmlogverify
atrshmlogdelete        atrshmlogoff.o      atrshmlogverify.c
atrshmlogdelete.c      atrshmlogon        atrshmlogverify.o
atrshmlogdelete.o      atrshmlogon.c      d1
atrshmlogdump          atrshmlogon.o      Doxyfile
atrshmlogdump.c        atrshmlogreader   files.txt
atrshmlogdump.o        atrshmlogreader.c  impls
atrshmlogerror         atrshmlogreaderd   libatrshmlog.a
atrshmlogfinish        atrshmlogreaderd.c atrshmlogreaderd.o
atrshmlogfinish.c      atrshmlogreset    atrshmlogreset
atrshmlogfinish.o      atrshmlogreset.c  atrshmlogreset.o
atrshmlog.h            atrshmlogsignalreader atrshmlogsignalreader
atrshmloginit           atrshmlogsignalreader.c atrshmlogsignalreader.o
atrshmloginit.c         atrshmloginternal atrshmlogsignalwriter
atrshmloginit.o         atrshmloginternal.c atrshmlogsignalwriter.c
atrshmloginternal      atrshmloginternal.h atrshmlogsignalwriter.o
atrshmloginternal.o    atrshmloginternal.o atrshmlogsort
atrshmlog.o             atrshmlogstat     atrshmlogstopreader
atrshmlogstopreader
```

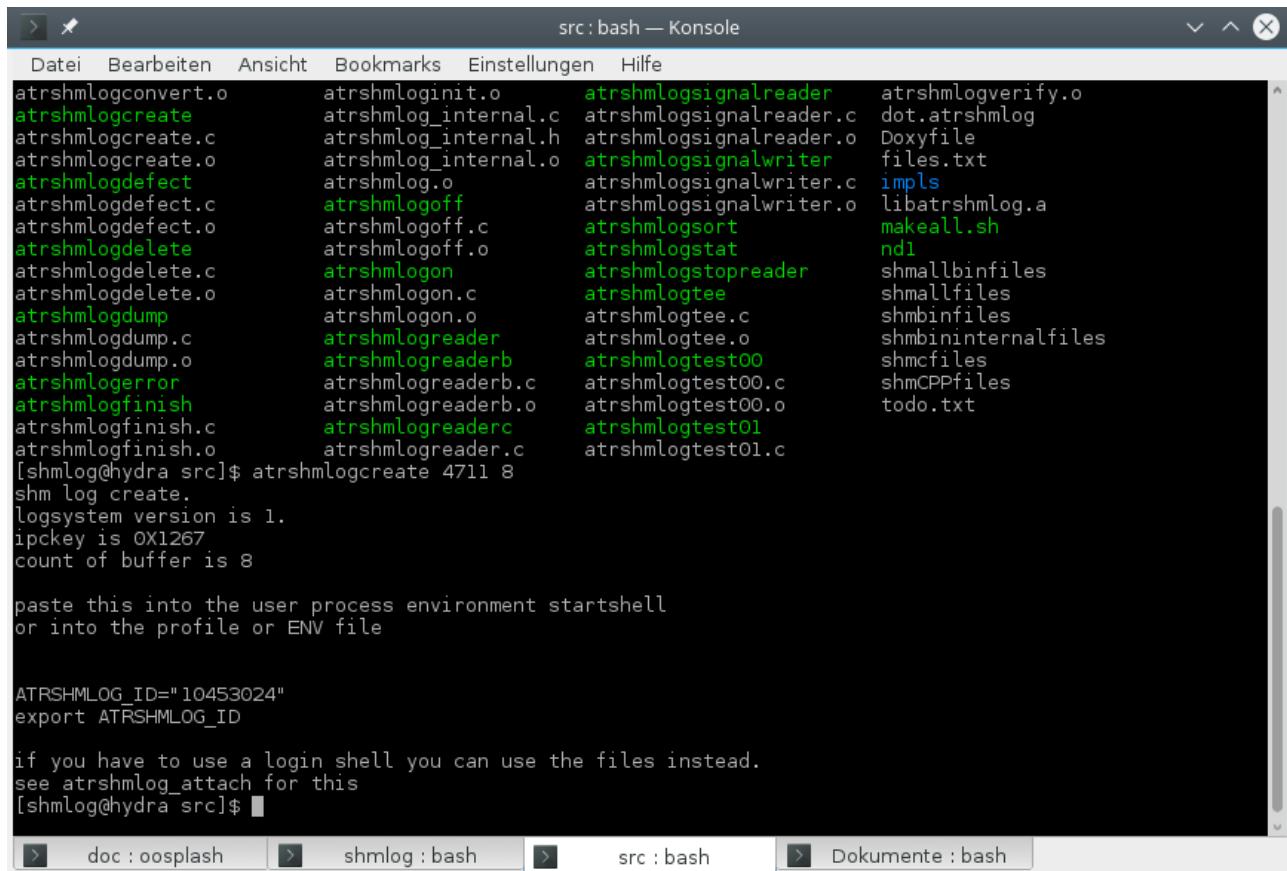
Illustration 9: Checking the build result with a ls

Looks fine for me, so change to the BASEDIR, then to doc, and check for the HTML folder. If the files are brand new and your browser is happy with the index.html that's it for the build.

See the testing for how to use them in detail. For a quick run we do it here just to have the first expression how to do things.

## First test : create the buffer

We create a shared memory buffer with the ipc key 4711 – that's a very well known number for a city nearby, so I think only some guru guys would try to use it in the system before. If you are having the right to do the shared memory thing, then it should work. So we start up the atrshmlogcreate and use the 4711 as key. The second parameter is for the number of buffers, we use an 8 here, which is for normal tests more than sufficient.



```
src : bash — Konsole
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
atrshmlogconvert.o atrshmloginit.o atrshmlogsignalreader atrshmlogverify.o
atrshmlogcreate atrshmlog_internal.c atrshmlogsignalreader.c dot.atrshmlog
atrshmlogcreate.c atrshmlog_internal.h atrshmlogsignalreader.o Doxyfile
atrshmlogcreate.o atrshmlog_internal.o atrshmlogsignalwriter files.txt
atrshmlogdefect atrshmlog.o atrshmlogsignalwriter.c impls
atrshmlogdefect.c atrshmloff atrshmlogsignalwriter.o libatrshmlog.a
atrshmlogdefect.o atrshmloff.c atrshmlogsort makeall.sh
atrshmlogdelete atrshmloff.o atrshmlogstat ndl
atrshmlogdelete.c atrshmlogon atrshmlogstopreader shmallbinfiles
atrshmlogdelete.o atrshmlogon.c atrshmlogtee shmallfiles
atrshmlogdump atrshmlogon.o atrshmlogtee.c shmbinfiles
atrshmlogdump.c atrshmlogreader atrshmlogtest00 shmbininternalfiles
atrshmlogdump.o atrshmlogreaderb atrshmlogtest00.c shmcfiles
atrshmlogerror atrshmlogreaderb.c atrshmlogtest00.o shmCPPfiles
atrshmlogfinish atrshmlogreaderb.o atrshmlogtest01 todo.txt
atrshmlogfinish.c atrshmlogreaderc atrshmlogtest01
atrshmlogfinish.o atrshmlogreader.c
[shmlog@hydra src]$ atrshmlogcreate 4711 8
shm log create.
logsystem version is 1.
ipckey is 0X1267
count of buffer is 8

paste this into the user process environment startshell
or into the profile or ENV file

ATRSHMLOG_ID="10453024"
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
[shmlog@hydra src]$
```

Illustration 10: A first test. We create a buffer for 8 log buffers with key 4711.

Looks good. The thing seems to make its job, there is an output that claims the ID should be in my case

10453024

and that I should set some environment variable that way. Well, cut and paste is easy for Linux, so I set it.

Side note: The last version of atrshmlogcreate also produces a file with the setting, its named

dot.attrshmlog and you can use it right away as a dot file in your shell. For mingw its called atrshmlog.cmd. And if you insist you can give the atrshmlogcreate a third parameter now with the name.

## First test : init the area

So now we have the raw buffer, its time to initialize it. We call atrshmloginit for this. The parameter is the number of buffers, in our case 8. And here it goes.

The screenshot shows a terminal window titled "src : bash — Konsole". The terminal displays the following command-line session:

```
src : bash — Konsole
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
attrshmlogdefect attrshmlog.o attrshmlogsignalwriter.c impls
attrshmlogdefect.c attrshmlogoff attrshmlogsignalwriter.o libattrshmlog.a
attrshmlogdefect.o attrshmlogoff.c attrshmlogsort makeall.sh
attrshmlogdelete attrshmlogoff.o attrshmlogstat ndi
attrshmlogdelete.c attrshmlogon attrshmlogstopreader shmallbinfiles
attrshmlogdelete.o attrshmlogon.c attrshmlogtee shmallfiles
attrshmlogdump attrshmlogon.o attrshmlogtee.c shmbinfiles
attrshmlogdump.c attrshmlogreader attrshmlogtee.o shmbininternalfiles
attrshmlogdump.o attrshmlogreaderb attrshmlogtest00 shmcfiles
attrshmlogerror attrshmlogreaderb.c attrshmlogtest00.c shmCPPfiles
attrshmlogfinish attrshmlogreaderb.o attrshmlogtest00.o todo.txt
attrshmlogfinish.c attrshmlogreaderc attrshmlogtest01
attrshmlogfinish.o attrshmlogreader.c attrshmlogtest01.c
[shmlog@hydra src]$ atrshmlogcreate 4711 8
shm log create.
logsystem version is 1.
ipckey is 0X1267
count of buffer is 8

paste this into the user process environment startshell
or into the profile or ENV file

ATRSHMLOG_ID="10846240"
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
[shmlog@hydra src]$ . dot.attrshmlog
[shmlog@hydra src]$ atrshmloginit 8
shm log attach and init.
logsystem version is 1.
[shmlog@hydra src]$
```

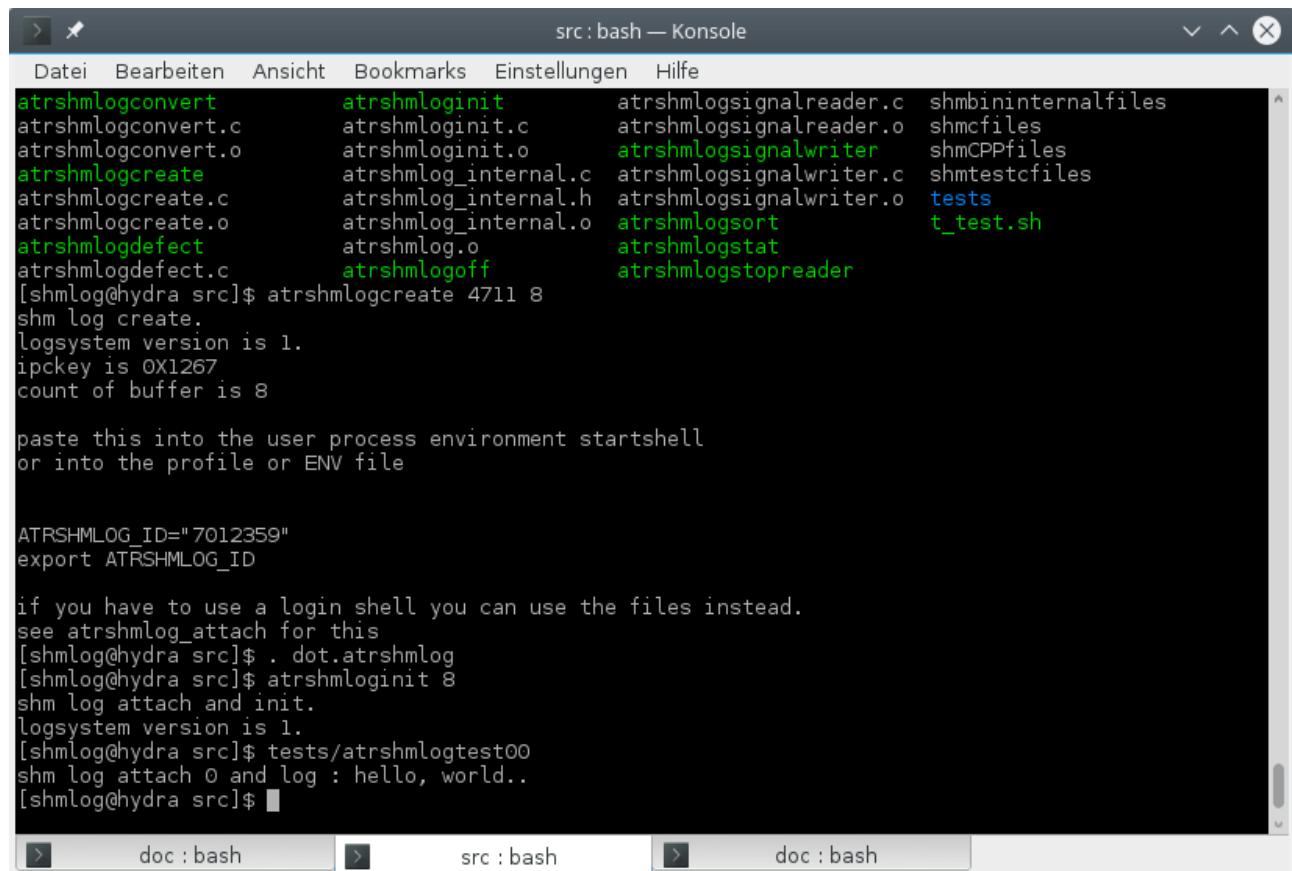
The terminal window has tabs at the bottom: "doc : oosplash", "shmlog : bash", "src : bash" (which is active), and "Dokumente : bash".

Illustration 11: Initialize the area

Setting the environment variable according to the output of the create and then initializing the buffer. In the last version the buffer count is already set in the environment and so we even don't need that here any more.

## First test: run the simplest test program

Nice. We have now the area. We could now try a verify, or a defect, or any other thing that uses it. So we begin for starters with the simplest program that can use the log, the hello world example. Its the atrshmlogtest00 . No parameters needed, simply start it.



The screenshot shows a terminal window titled "src : bash — Konsole". The window contains the following text:

```
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
atrshmlogconvert atrshmloginit atrshmlogsignalreader.c shmbininternalfiles
atrshmlogconvert.c atrshmloginit.c atrshmlogsignalreader.o shmcfiles
atrshmlogconvert.o atrshmloginit.o atrshmlogsignalwriter atrshmlogsignalwriter.c shmcPPfiles
atrshmlogcreate atrshmlog_internal.c atrshmlogsignalwriter.o shmtestcfiles
atrshmlogcreate.c atrshmlog_internal.h atrshmlogsort tests
atrshmlogcreate.o atrshmlog_internal.o atrshmlogstat t_test.sh
atrshmlogdefect atrshmlog.o atrshmlogstopreader
atrshmlogdefect.c atrshmlogoff

[shmlog@hydra src]$ atrshmlogcreate 4711 8
shm log create.
logsystem version is 1.
ipckey is 0X1267
count of buffer is 8

paste this into the user process environment startshell
or into the profile or ENV file

ATRSHMLOG_ID="7012359"
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
[shmlog@hydra src]$ . dot.atrshmlog
[shmlog@hydra src]$ atrshmloginit 8
shm log attach and init.
logsystem version is 1.
[shmlog@hydra src]$ tests/atrshmlogtest00
shm log attach 0 and log : hello, world..
[shmlog@hydra src]$
```

Illustration 12: Testing the log with the hello world demo program.

Looks good.

Now we have to check if we really have log.

## **First test: getting the log into files**

The thing can be done by using verify, which will give us at least one used buffer, or we can use dump, and then check the hex codes ( ups, I think I pass that) or we can use the reader to fetch the log and write it down.

For the simplicity of the first steps we use the reader. Its after all the usual way to get the log. And a verify is nice – only one line to type, but – but shown not the content.

OK. So the reader it is.

In this version the reader will be the atrshmlogreaderd .

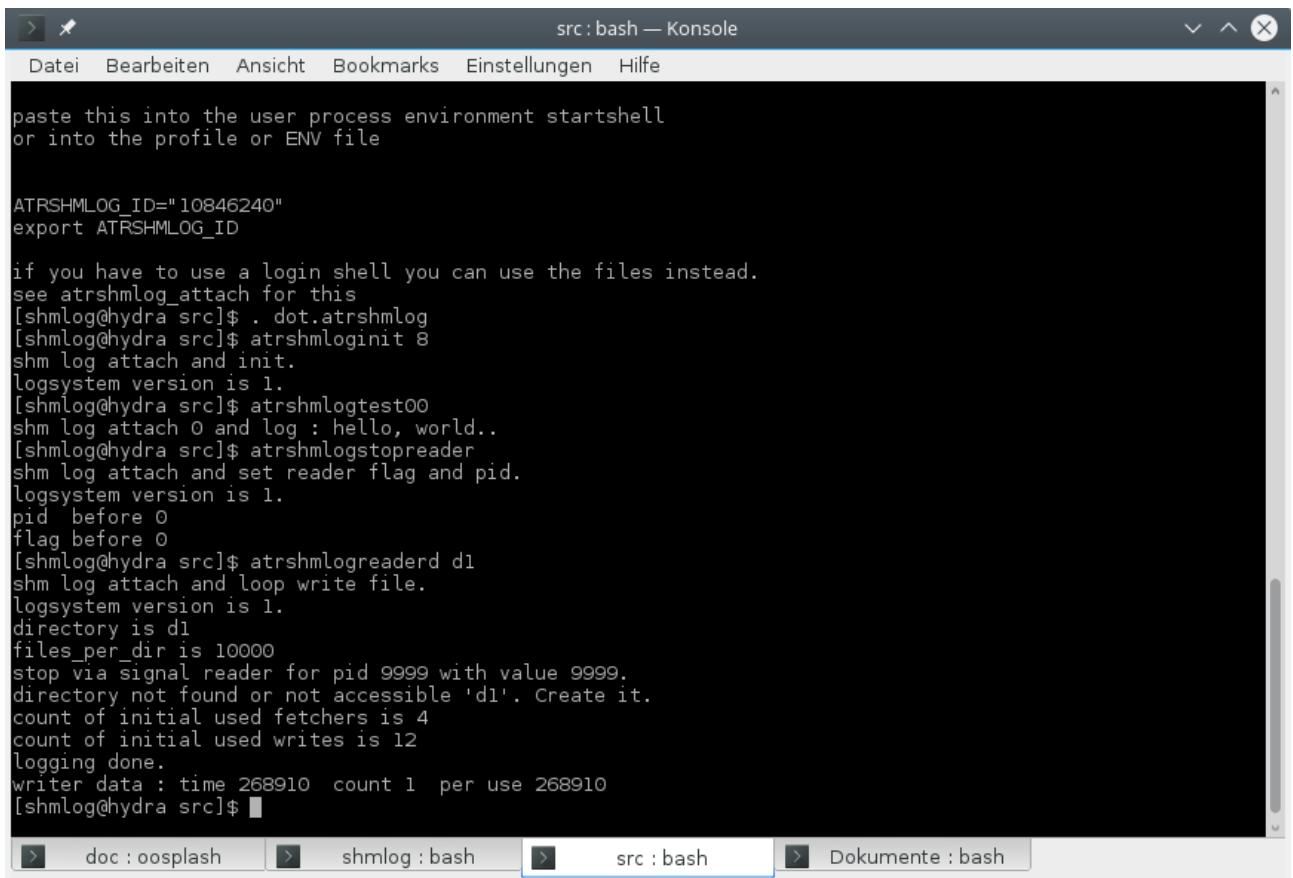
And we have one complication. The thing will start, connect to the area, then check for data to transfer, do the transfer, and then sleep as long as we don't stop it and wait for more data.

So we can do two things here. Start it, wait for a small time, lets say 5 seconds, then kill it with the usual ctrl-c thing. Second thing is to start and then signal the reader its done. In this case the readerd will transfer the buffer and end after some seconds by itself. This can even be set in advance of the start of the reader, so it simply make the transfer for the buffers in place and ends immediate after that.

We use the later thing, because we can do it without that ctrl-c thing, and then see for the output. To do this we have to set the signal in place with atrshmlogstopreader. No parameters needed.

Then we can start the reader. It needs a target directory, so I give it a d1 as target. The rest is done by the reader.

This looks like this for me



The screenshot shows a terminal window titled "src : bash — Konsole". The window contains the following text:

```
paste this into the user process environment startshell  
or into the profile or ENV file

ATRSHMLOG_ID="10846240"  
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.  
see atrshmlog_attach for this  
[shmlog@hydra src]$ . dot.atrshmlog  
[shmlog@hydra src]$ atrshmloginit 8  
shm log attach and init.  
logsystem version is 1.  
[shmlog@hydra src]$ atrshmlogtest00  
shm log attach 0 and log : hello, world..  
[shmlog@hydra src]$ atrshmlogstopreader  
shm log attach and set reader flag and pid.  
logsystem version is 1.  
pid before 0  
flag before 0  
[shmlog@hydra src]$ atrshmlogreaderd d1  
shm log attach and loop write file.  
logsystem version is 1.  
directory is d1  
files_per_dir is 10000  
stop via signal reader for pid 9999 with value 9999.  
directory not found or not accessible 'd1'. Create it.  
count of initial used fetchers is 4  
count of initial used writes is 12  
logging done.  
writer data : time 268910 count 1 per use 268910  
[shmlog@hydra src]$ █
```

The terminal has four tabs at the bottom: "doc : oosplash", "shmlog : bash", "src : bash" (which is active), and "Dokumente : bash".

*Illustration 13: Setting the area to no more logging and reading the data.*

Looking good again.

The reader took some time to write down the buffer. Its also the average time, because there was only one buffer. We use those numbers later in the adjustment. The reader also told us that it uses multiple threads for doing its stuff. See the reader chapter later about this and the adjustment chapter, too.

For now we only have to check if the log is really in the file system. We can do this with an ls. To be not to fast : its in the first directory that the reader uses. That's directory d1/0 in this case. The reader simply starts to create directories in the root, that's d1 and uses numbers for the name – if your OS cannot do this change the reader to use something like s0 or similar.

The screenshot shows a terminal window titled "src : bash — Konsole". The menu bar includes "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". The terminal content displays the configuration and execution of the atrshmlog tool. It starts with setting the environment variable ATRSHMLOG\_ID to "10846240" and then executing atrshmlog. The log output indicates the creation of a log file named "shmlog" with a size of 554 bytes. The terminal then lists the contents of the directory "d1" using the command "ls -l d1/0/atrshmlog\_p22115\_t22115\_f0.bin", showing a single file "shmlog" with a timestamp of 27. Okt 15:10.

```
ATRSHMLOG_ID="10846240"
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
[shmlog@hydra src]$ . dot.atrshmlog
[shmlog@hydra src]$ atrshmloginit 8
shm log attach and init.
logsystem version is 1.
[shmlog@hydra src]$ atrshmlogtest00
shm log attach 0 and log : hello, world..
[shmlog@hydra src]$ atrshmlogstopreader
shm log attach and set reader flag and pid.
logsystem version is 1.
pid before 0
flag before 0
[shmlog@hydra src]$ atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
logging done.
writer data : time 268910 count 1 per use 268910
[shmlog@hydra src]$ ls -l d1/0/atrshmlog_p22115_t22115_f0.bin
-rw----- 1 shmlog shmlog 554 27. Okt 15:10 d1/0/atrshmlog_p22115_t22115_f0.bin
[shmlog@hydra src]$
```

*Illustration 14: The content of the directory tree d1 after transfer.*

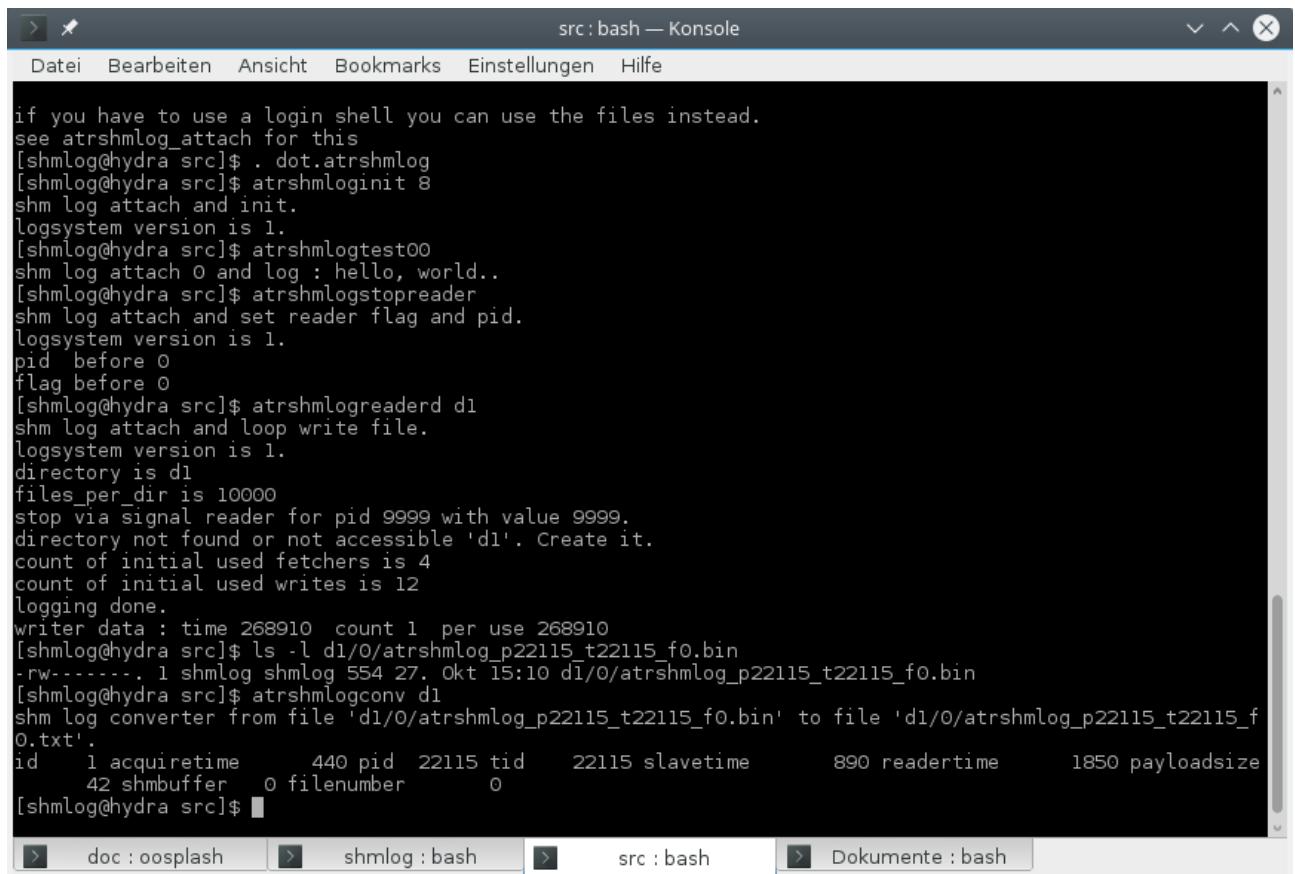
OK. There it is. The numbers depend on the process id and tid, so they change normally.

Now we have to use the converter. To do that is simple. Start it – its not connected to an area, so you can use it at any time, only two parameters and that's it.

The other way is to use the helper atrshmlogconv. Calls the converter for every transferred bin file in the tree.

## First time : converting binary to human readable

That's the way we do it normally. So here it is



The screenshot shows a terminal window titled "src : bash — Konsole". The window contains a command-line session demonstrating the conversion of a binary log file to human-readable text. The session starts with a message about using a login shell instead of files. It then shows the use of "atrshmlog\_attach" and "atrshmloginit" commands to initialize a shared memory log. Subsequent commands like "atrshmlogtest00" and "atrshmlogstopreader" are used to test and stop the log reader. The "atrshmlogreaderd" command is run to start a reader thread. The log displays various statistics such as writer data, file counts, and payload sizes. Finally, the "atrshmlogconv" command is used to convert a binary file ("d1/0/atrshmlog\_p22115\_t22115\_f0.bin") into a text file ("d1/0/atrshmlog\_p22115\_t22115\_f0.txt"). The terminal also lists other tabs like "doc : oosplash" and "shmlog : bash".

```
if you have to use a login shell you can use the files instead.  
see atrshmlog_attach for this  
[shmlog@hydra src]$ . dot.atrshmlog  
[shmlog@hydra src]$ atrshmloginit 8  
shm log attach and init.  
logsystem version is 1.  
[shmlog@hydra src]$ atrshmlogtest00  
shm log attach 0 and log : hello, world..  
[shmlog@hydra src]$ atrshmlogstopreader  
shm log attach and set reader flag and pid.  
logsystem version is 1.  
pid before 0  
flag before 0  
[shmlog@hydra src]$ atrshmlogreaderd d1  
shm log attach and loop write file.  
logsystem version is 1.  
directory is d1  
files_per_dir is 10000  
stop via signal reader for pid 9999 with value 9999.  
directory not found or not accessible 'd1'. Create it.  
count of initial used fetchers is 4  
count of initial used writes is 12  
logging done.  
writer data : time 268910 count 1 per use 268910  
[shmlog@hydra src]$ ls -l d1/0/atrshmlog_p22115_t22115_f0.bin  
-rw----- 1 shmlog shmlog 554 27. Okt 15:10 d1/0/atrshmlog_p22115_t22115_f0.bin  
[shmlog@hydra src]$ atrshmlogconv d1  
shm log converter from file 'd1/0/atrshmlog_p22115_t22115_f0.bin' to file 'd1/0/atrshmlog_p22115_t22115_f0.txt'.  
id 1 acquiretime 440 pid 22115 tid 22115 slavetime 890 readertime 1850 payloadsiz  
42 shmbuffer 0 filenumber 0  
[shmlog@hydra src]$
```

Illustration 15: Convert binary to human readable text.

Converting all bin files in the tree with the helper `atrshmlogconv` , and some new statistics.

Looks good. We have some interesting info's in the second line, but for now let them be there. Its part of the adjustment process to use these numbers.

We have now a converted bin file into a readable text. So here is the content. We use the simple cat.

The screenshot shows a terminal window titled "src : bash — Konsole". The terminal displays the output of several commands related to the "atrshmlog" system. The commands include ". dot.atrshmlog", ".atrshmloginit 8", "shm log attach and init.", "logsystem version is 1.", ".atrshmlogtest00", "shm log attach 0 and log : hello, world..", ".atrshmlogstopreader", "shm log attach and set reader flag and pid.", "logsystem version is 1.", "pid before 0", "flag before 0", ".atrshmlogreaderd d1", "shm log attach and loop write file.", "logsystem version is 1.", "directory is d1", "files\_per\_dir is 10000", "stop via signal reader for pid 9999 with value 9999.", "directory not found or not accessible 'd1'. Create it.", "count of initial used fetchers is 4", "count of initial used writes is 12", "logging done.", "writer data : time 268910 count 1 per use 268910", "[shmlog@hydra src]\$ ls -l d1/0/atrshmlog\_p22115\_t22115\_f0.bin", "-rw----- 1 shmlog shmlog 554 27. Okt 15:10 d1/0/atrshmlog\_p22115\_t22115\_f0.bin", "[shmlog@hydra src]\$ atrshmlogconv d1", "shm log converter from file 'd1/0/atrshmlog\_p22115\_t22115\_f0.bin' to file 'd1/0/atrshmlog\_p22115\_t22115\_f0.txt'.", "id 1 acquiretime 440 pid 22115 tid 22115 slavetime 890 readertime 1850 payloadszie", "42 shmbuffer 0 filenumber 0", "[shmlog@hydra src]\$ cat d1/0/atrshmlog\_p22115\_t22115\_f0.txt", "0000022115 000000000022115 000 0000000000000000 00004129189524110 00004129189524110 000000000000000000 00 1477573510525365891 1477573510525365891 000000000000000000 000000000001 P 0000000001 hello, world.", "[shmlog@hydra src]\$". Below the terminal window, there is a tab bar with four tabs: "doc : oosplash", "shmlog : bash", "src : bash", and "Dokumente : bash".

*Illustration 16: The resulting log in text form.*

That's a lot of numbers for now, and the text at the end.

Seems to me it worked.

## When it does not work for you

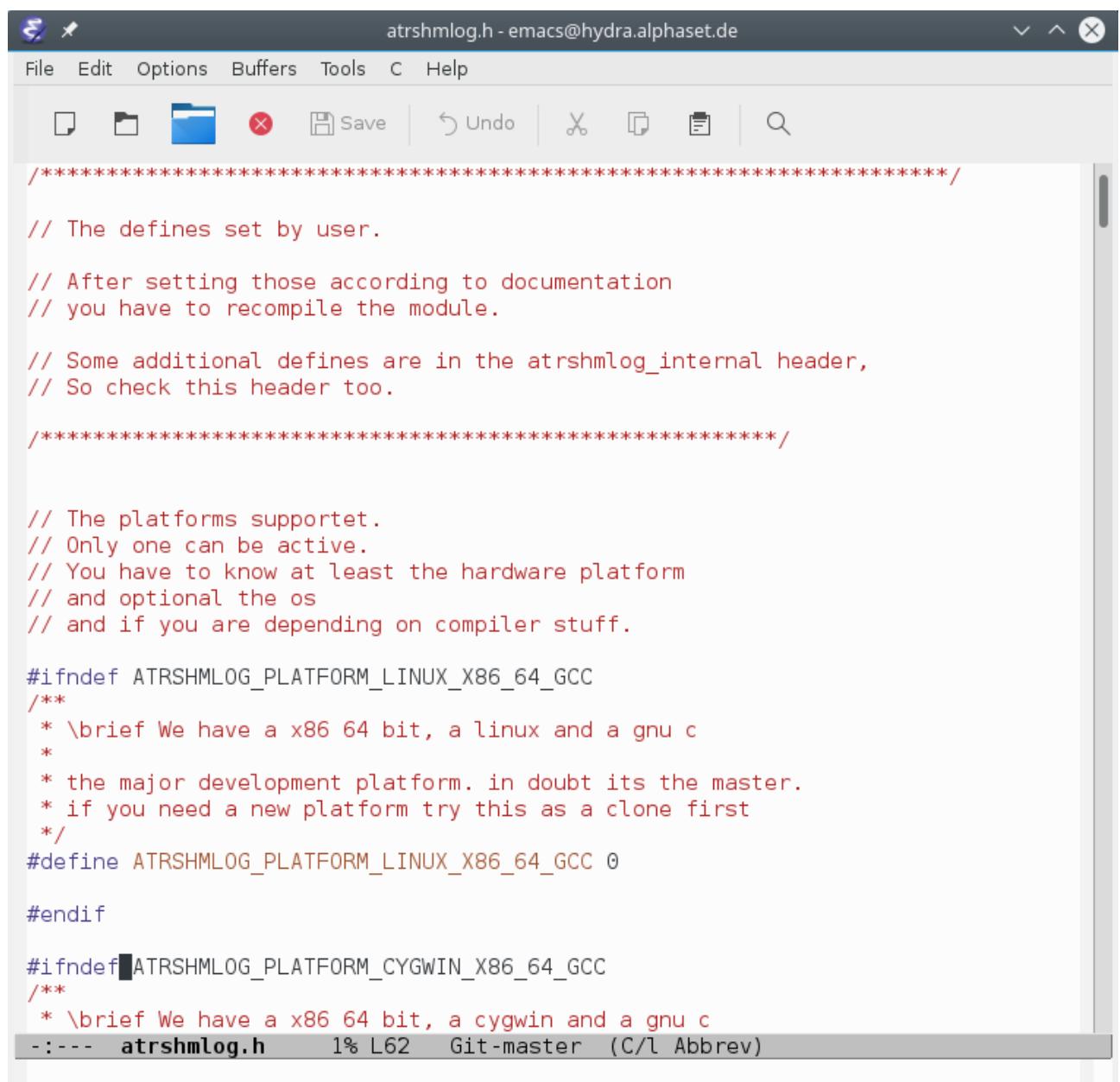
OK. Bad things can happen.

### Is it the wrong platform ?

First check if you are really on a supported platform.

Check the dot files in BASEDIR for this.

Check the header atrshmlog.h for it:



The screenshot shows an Emacs window with the title "atrshmlog.h - emacs@hydra.alphaset.de". The buffer contains C code for the atrshmlog.h header. The code includes comments about supported platforms and compiler-specific defines. It defines two macros: ATRSHMLOG\_PLATFORM\_LINUX\_X86\_64\_GCC and ATRSHMLOG\_PLATFORM\_CYGWIN\_X86\_64\_GCC. The code is color-coded, with comments in red and preprocessor directives in blue.

```
*****  
// The defines set by user.  
  
// After setting those according to documentation  
// you have to recompile the module.  
  
// Some additional defines are in the atrshmlog_internal header,  
// So check this header too.  
*****  
  
// The platforms supportet.  
// Only one can be active.  
// You have to know at least the hardware platform  
// and optional the os  
// and if you are depending on compiler stuff.  
  
#ifndef ATRSHMLOG_PLATFORM_LINUX_X86_64_GCC  
/**  
 * \brief We have a x86 64 bit, a linux and a gnu c  
 *  
 * the major development platform. in doubt its the master.  
 * if you need a new platform try this as a clone first  
 */  
#define ATRSHMLOG_PLATFORM_LINUX_X86_64_GCC 0  
  
#endif  
  
#ifndef ATRSHMLOG_PLATFORM_CYGWIN_X86_64_GCC  
/**  
 * \brief We have a x86 64 bit, a cygwin and a gnu c  
----- atrshmlog.h 1% L62 Git-master (C/l Abbrev)
```

Illustration 17: Supported platforms in the atrshmlog.h header

There is a small helper to do the check, for now we do it manual. Or use the atrshmlogchecksystem.

So you will find as supported platform a define. And the define has a value 1 or 0. There can be only one of them on 1, so check which one it is. If all are 0 check if your system is in the supported platforms with atrshmlogchecksystem and with the dot files in BASEDIR.

For the vanilla module it is the

ATRSHMLOG\_PLATFORM\_LINUX\_X86\_64\_GCC

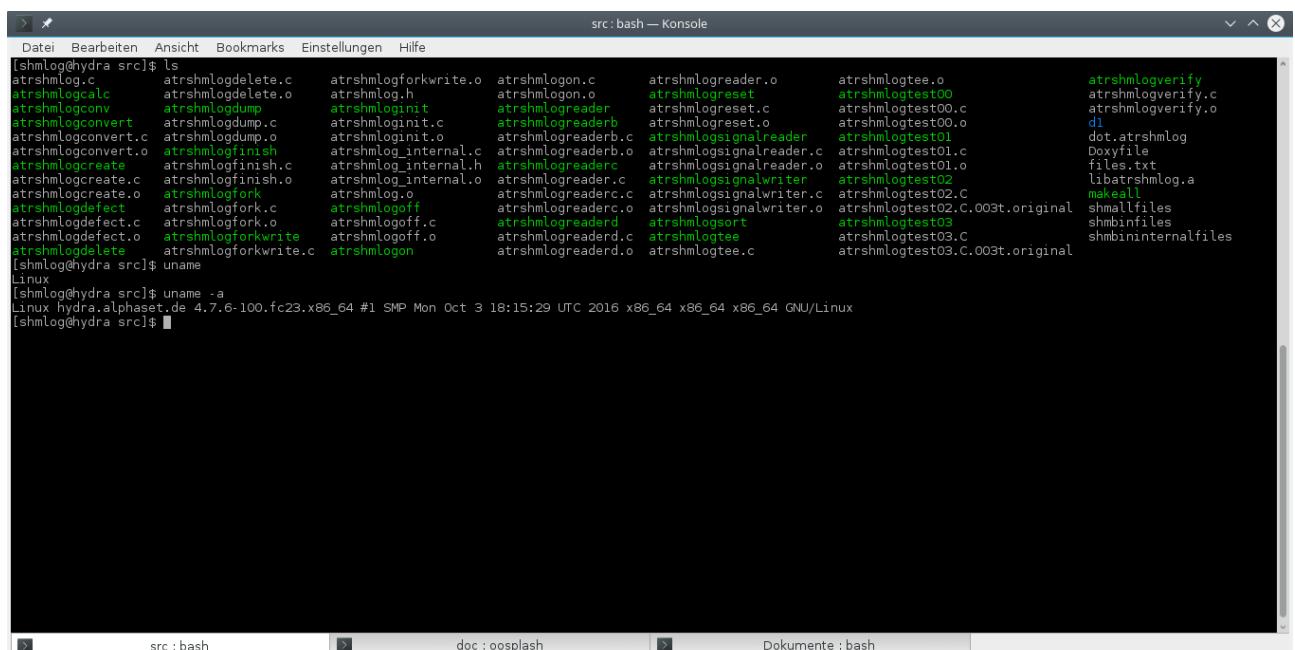
So if you have another platform you have to check for it and then change the 1 to 0 and for that platform the 0 to 1. Use the vanilla dot.platform.sh.

Then try again the makeall.sh.

If you don't know your platform its a bit harder. First there is the question if you are on an OS that is supported. For this version the supported OS are:

- Linux
- Cygwin
- Mingw (via cygwin)
- BSD
- Solaris

If you are on a Linux but does not know it, you can find out with the uname program. Its installed on nearly every Linux and also on nearly every other UNIX and UNIX like OS.



```
[shmLog@hydra src]$ ls
atrhsnLog.c      atrshmlogdelete.c    atrshmlogforkwrite.o  atrshmlogon.c      atrshmlogreader.o   atrshmlogtee.o      atrshmlogverify
atrhsnLogcalc.o  atrshmlogdelete.o    atrshmlog.h        atrshmlogon.o      atrshmlogreset     atrshmlogtest00
atrhsnLogconv.o  atrshmlogdump       atrshmloginit       atrshmlogreader    atrshmlogreset.c   atrshmlogtest00.c
atrhsnLogconvert.o atrshmlogdump.c    atrshmloginit.c    atrshmlogreaderb   atrshmlogreset.o   atrshmlogtest00.o
atrhsnLogconvert.c atrshmlogdump.o   atrshmloginit.o    atrshmlogreaderb.c  atrshmlogreset.b   atrshmlogtest01
atrhsnLogconvert.o atrshmlogfinish   atrshmlog_internal.c atrshmlogreaderb.o  atrshmlogreset.b.c atrshmlogtest01.c
atrhsnLogcreate   atrshmlogfinish.c  atrshmlog_internal.h atrshmlogreaderc   atrshmlogreset.c   atrshmlogtest01.o
atrhsnLogcreate.c atrshmlogfinish.o  atrshmlog_internal.o atrshmlogreaderc.c  atrshmlogreset.c   atrshmlogtest01.o
atrhsnLogcreate.o atrshmlogfork     atrshmlog.h        atrshmlogreaderc.c  atrshmlogreset.c   atrshmlogtest02
atrhsnLogdefect   atrshmlogfork.c    atrshmlogoff       atrshmlogreaderc.o  atrshmlogreset.c   atrshmlogtest02
atrhsnLogdefect.c atrshmlogfork.o   atrshmlogoff.c    atrshmlogreaderd   atrshmlogsort     atrshmlogtest03
atrhsnLogdefect.o atrshmlogforkwrite atrshmlogoff.o    atrshmlogreaderd.c  atrshmlogtee     atrshmlogtest03.C
atrhsnLogdelete   atrshmlogforkwrite.c atrshmlogon       atrshmlogreaderd.o  atrshmlogtee.c    atrshmlogtest03.C.003t.original
[shmLog@hydra src]$ uname
Linux
[shmLog@hydra src]$ uname -a
Linux hydra.alphaset.de 4.7.6-100.fc23.x86_64 #1 SMP Mon Oct 3 18:15:29 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
[shmLog@hydra src]$
```

Illustration 18: Detecting the OS and the architecture with uname

OK. This gives you at least a clue about the OS. The atrshmlogchecksystem can also check for the

supported platforms what it is exactly. It uses for this some heuristics about the etc directory for the Linux and BSD systems.

When you get also the -a parameter to work you normally also get the architecture of the system.

Here is an output for another system:

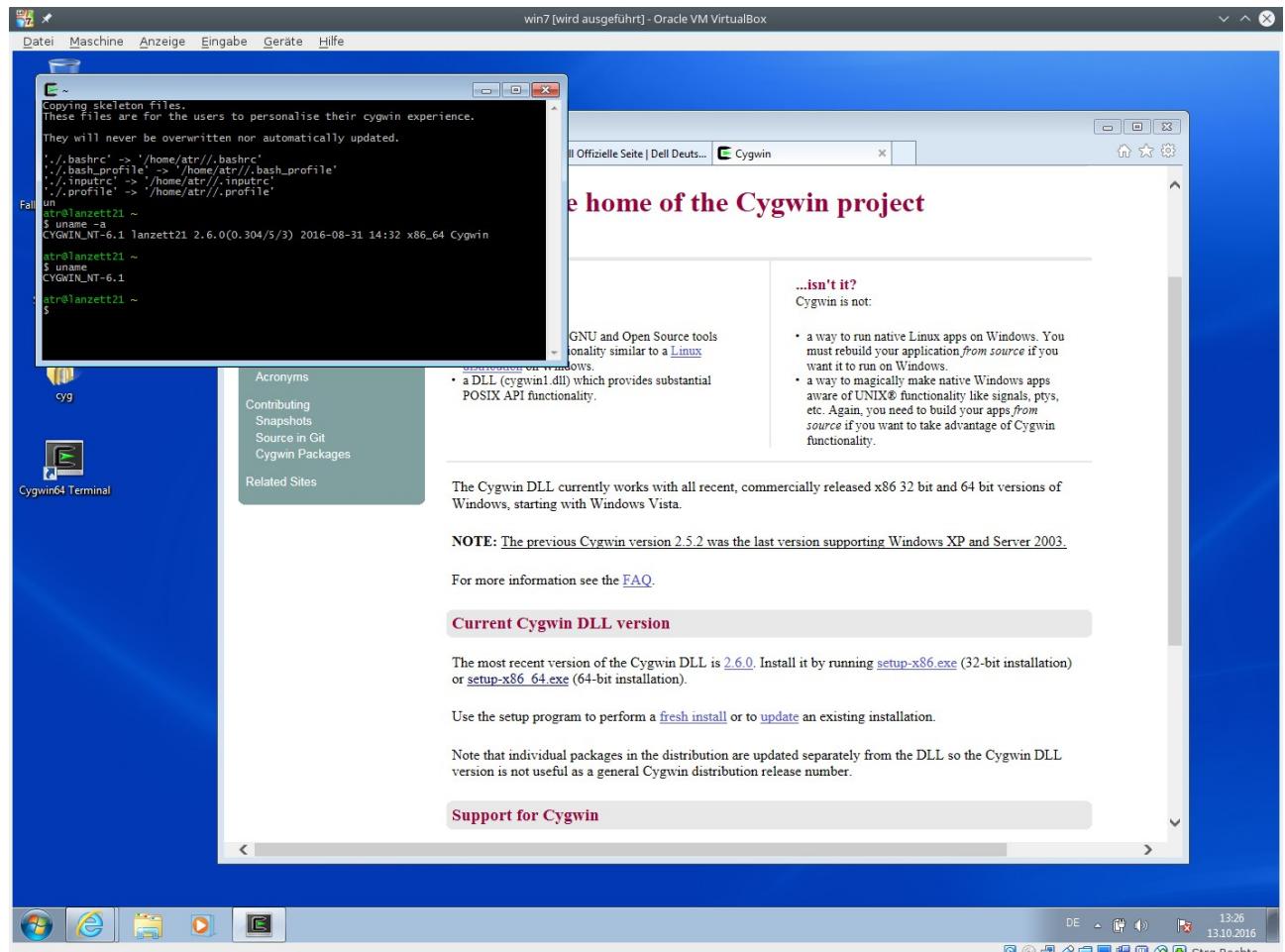


Illustration 19: Platform check for another system.

And here is a cygwin for starters.

## Check the C compiler

Next comes the question of the installed compiler and tool chain.

You will need at least an ISO C 11 conform compiler with support for atomics.

For the compiler you have to check the documentation. For my platform I have to give it a -std=gnu11 flag to do the job.

A quick check is to compile a program that includes stdatomic.h . So we simply try a five liner:

```
#include <stdio.h>
#include <stdatomic.h>
int main ()
{
    printf(„hello world\n“);
}
```

or simply use the atrshmlogcheckc.c file as it is done by atrshmlogchecksyst in tests subdirectory.

And that's it:

The screenshot shows a terminal window titled "src : bash — Konsole". The terminal displays the following command-line session:

```
[shmLog@hydra src]$ ls
[shmLog@hydra src]$ atrshmlog.c atrshmlogdelete.c atrshmlogforkwrite.o atrshmlogon.c atrshmlogreader.o atrshmlogtee.o atrshmlogverify
atrshmlogcalc atrshmlogdelete.o atrshmlog.h atrshmlogon.o atrshmlogreset atrshmlogtest00 atrshmlogverify.c
atrshmlogconv atrshmlogdump atrshmloginit atrshmlogreader atrshmlogreset.c atrshmlogtest00.c
atrshmlogconvert atrshmlogdump.c atrshmloginit.c atrshmlogreaderb atrshmlogreset.o atrshmlogtest00.o
atrshmlogconvert.c atrshmlogdump.o atrshmloginit.o atrshmlogreaderb.c atrshmlogsignalreader atrshmlogtest01
atrshmlogcreate atrshmlogfinish atrshmlog_internal.c atrshmlogreaderb.o atrshmlogsignalreader.c atrshmlogtest01.c
atrshmlogcreate.c atrshmlogfinish.o atrshmlog_internal.h atrshmlogreaderc atrshmlogsignalreader.o atrshmlogtest01.o
atrshmlogcreate.o atrshmlogfork atrshmlog_internal.o atrshmlogreader.c atrshmlogsignalwriter atrshmlogtest02
atrshmlogcreate.o atrshmlogfork atrshmlogfork.c atrshmlogoff atrshmlogreaderc.o atrshmlogsignalwriter.o atrshmlogtest02.C
atrshmlogdefect atrshmlogfork.o atrshmlogfork.f atrshmlogoff.c atrshmlogreaderd atrshmlogsort atrshmlogtest03
atrshmlogdefect.c atrshmlogfork.c atrshmlogfork.o atrshmlogoff.o atrshmlogreaderd.c atrshmlogtee atrshmlogtest03.C
atrshmlogdelete atrshmlogforkwrite atrshmlogon atrshmlogreaderd.o atrshmlogreaderd.o atrshmlogtee.o atrshmlogtest03.C.003t.original
[shmLog@hydra src]$ uname
Linux
[shmLog@hydra src]$ uname -a
Linux hydra.alphaset.de 4.7.6-100.fc23.x86_64 #1 SMP Mon Oct 3 18:15:29 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
[shmLog@hydra src]$ cat - > h.c
#include <stdio.h>
#include <stdatomic.h>
int main ()
{
    printf("hello world\n");
}
[shmLog@hydra src]$ cc h.c
[shmLog@hydra src]$ ./a.out
hello world
[shmLog@hydra src]$
```

Illustration 20: Check for a conform compiler and the atomic header.

OK, the header is there and the compiler seems at least not complain about it.

So next is a test for thread local, which is new in C 11.

The screenshot shows a terminal window titled "src : bash – Konsole". The terminal displays the following command-line session:

```
[shmlog@hydra src]$ ls
atrshmLog.c atrshmLogDelete.c atrshmLogfwrite.o atrshmLogon.c atrshmLogReader.o atrshmLogtee.o atrshmLogverify
atrshmLogCalc atrshmLogDelete.o atrshmLog.h atrshmLogon.o atrshmLogReset atrshmLogTest00
atrshmLogConv atrshmLogDump atrshmLogInit atrshmLogReader atrshmLogReset.c atrshmLogTest00.o atrshmLogVerify.c
atrshmLogConvert atrshmLogDump.c atrshmLogInit.c atrshmLogReaderb atrshmLogReset.o atrshmLogTest00.o atrshmLogVerify.o
atrshmLogConvert.o atrshmLogDump.o atrshmLogInit.o atrshmLogReaderb.c atrshmLogSignalReader atrshmLogTest01
atrshmLogCreate atrshmLogFinish atrshmLogInternal.c atrshmLogReaderb.o atrshmLogSignalReader.c atrshmLogTest01.c
atrshmLogCreate atrshmLogFinish.c atrshmLogInternal.h atrshmLogReaderc atrshmLogSignalReader.o atrshmLogTest01.o
atrshmLogCreate.c atrshmLogFinish.o atrshmLogInternal.o atrshmLogReaderc atrshmLogSignalWriter atrshmLogTest02
atrshmLogCreate.o atrshmLogFork atrshmLogInternal.o atrshmLogReaderc.c atrshmLogSignalWriter.c atrshmLogTest02.C
atrshmLogDefect atrshmLogFork.c atrshmLogff atrshmLogReaderc.o atrshmLogSignalWriter.o atrshmLogTest02.C.003t.original
atrshmLogDefect.c atrshmLogFork.o atrshmLogff.c atrshmLogReaderd atrshmLogSort atrshmLogTest03
atrshmLogDelete atrshmLogForkWrite atrshmLogff.o atrshmLogReaderd.c atrshmLogTee atrshmLogTest03.C
atrshmLogDelete.o atrshmLogForkWrite.c atrshmLogon atrshmLogReaderd.o atrshmLogTee.o atrshmLogTest03.C.003t.original
Linux
[shmlog@hydra src]$ uname -a
Linux hydra.alphaset.de 4.7.6-100.fc23.x86_64 #1 SMP Mon Oct 3 18:15:29 UTC 2016 x86_64 x86_64 x86_64 GNU/Linux
[shmlog@hydra src]$ cat - > h.c
#include <stdio.h>
#include <stdatomic.h>
int main ()
{
printf("hello world\n");
}
[shmlog@hydra src]$ cc h.c
[shmlog@hydra src]$ ./a.out
hello world
[shmlog@hydra src]$ rm h.c a.out
[shmlog@hydra src]$ cat - > h.c
#include <stdio.h>
_Thread_local static int a = 0;
int main()
{
printf("hello world\n");
}
[shmlog@hydra src]$ cc h.c
[shmlog@hydra src]$ ./a.out
hello world
[shmlog@hydra src]$
```

Illustration 21: Check for a C 11 feature, the \_Thread\_local

If you have any trouble at this point, you have to check the compiler and the libraries. This is depending on your platform and so I can give you only a hint. Call the compiler with a flag -help or --help and see for its output, then check the man page or if it does not have man pages any doc in the /usr/share directories for documentation to the thing. Having this said – I assume you are running a UNIX box and it has at least standard layout for the files.

There is actually one supported platform that does not support thread local storage. It is the OpenBSD. Then you can resort to the thread specific storage as it is done there.

What if my compiler is indeed C 11 but no support for stdatomic ? That's for example the case for CentOS 7.2 and SLES 12 SP2 actually ( gcc 4.8 does the C11 thing, but no stdatomic.h there).

One thing possible: Install a newer version of the compiler (here its was done for gcc 5.4.0 and the source code, the thing is not available as rpm for now).

What if this does not work ?

Here is at least one project that tries to deliver for C11 compilers the atomics that don't have them on the net. Try to find this : <http://stdatomic.gforge.inria.fr>

If any thing fails its at least a try to get the gcc up and running. It supports the atomic till C11 was integrated from 4.9.1 on, so chances are good you get it with this. Then you have to check if the library is linkage compatible with your target and if that's the case you are back in business. This was done for example for Solaris 5.11 (now socalled opensolaris 11.3 x86 64 ).

Its also possible to use a clang 3.8 at least, if your platform supports that (see the FreeBSD and

NetBSD for this).

If you have made it for the C module and the library and the programs you can use it.

## **Check the C++ compiler**

The C++ programs are another story.

Its possible for the platform I use to use the C header with atomics and with inline mode. And the C++ compiler does it too. So we have a coupling here of the two things.

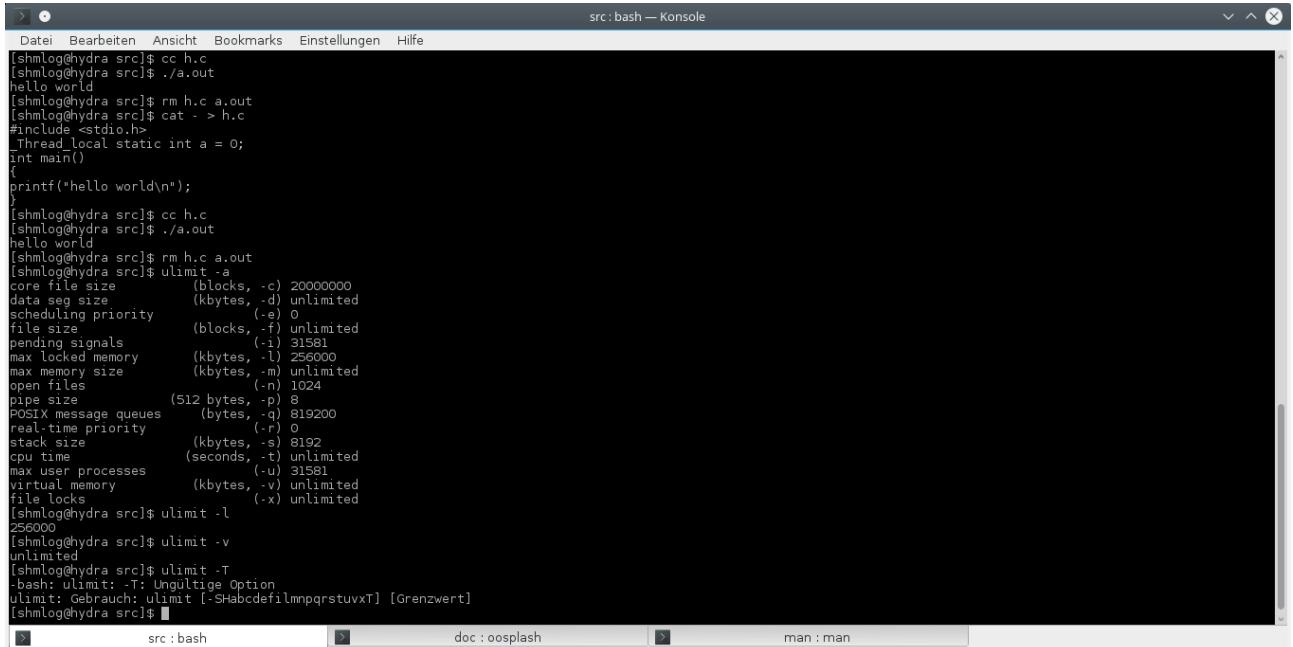
If your C compiler supports the inline that does not necessarily mean your C++ compiler agrees. Then you have to switch the inline off for the C++ part. And that means you need link able functions in the module too.

This is not supported for now, so you have to switch inline off for both.

If you don't want to do this there is a unsupported hack, its that you compile the module inline, and make a new file of the only inlined stuff, compile that without inline and link it with the C++ together. Its not tested but it should do the job – don't forget to use the right headers after this hack, you will need one with and one without inline in most cases.

## Check the OS if the things fail to run

Having it compiled can still mean you get trouble for the execution.



The screenshot shows a terminal window titled "src : bash — Konsole". The terminal displays the following command-line session:

```
[shnlog@hydra src]$ cc h.c
[shnlog@hydra src]$ ./a.out
hello world
[shnlog@hydra src]$ rm h.c a.out
[shnlog@hydra src]$ cat - > h.c
#include <stdio.h>
_Thread local static int a = 0;
int main()
{
printf("hello world\n");
}
[shnlog@hydra src]$ cc h.c
[shnlog@hydra src]$ ./a.out
hello world
[shnlog@hydra src]$ rm h.c a.out
[shnlog@hydra src]$ ulimit -a
core file size          (blocks, -c) 2000000
data seg size           (kbytes, -d) unlimited
scheduling priority      (-e) 0
file size               (blocks, -f) unlimited
pending signals          (-i) 31581
max locked memory       (kbytes, -l) 256000
max memory size         (kbytes, -m) unlimited
open files              (-n) 1024
pipe size                (512 bytes, -p) 8
POSIX message queues    (bytes, -q) 819200
real-time priority       (-r) 0
stack size               (kbytes, -s) 8192
cpu time                 (seconds, -t) unlimited
max user processes        (-u) 31581
virtual memory            (kbytes, -v) unlimited
file locks               (-x) unlimited
[shnlog@hydra src]$ ulimit -l
256000
[shnlog@hydra src]$ ulimit -v
unlimited
[shnlog@hydra src]$ ulimit -T
-bash: ulimit: -T: Ungültige Option
ulimit: Gebrauch: ulimit [-SHabcdefilmnpqrstuvwxyzT] [Grenzwert]
[shnlog@hydra src]$
```

Illustration 22: Limits for the user

We see here that there are some settings on my box that could affect the shared memory use.

I have used here ulimit, its a shell build.

If you encounter problems that would be the first thing to check for.

Second is that the OS allows only special users to build a shared memory buffer.

So you have to know the system for the settings then.

## **When it does not fit for your needs**

So far you have made it, and it made the tests too.

But you recognize you need something different.

So what can you do about it ?

### **Changes with the build in initialization stuff**

The first way is to simply check the possible changes for values the module uses in the already made initialization process. You have to check for the attach function. It contains a list for changeable values.

Here is short overview:

- `atrshmlog_buffer_strategy`  
You can set the wait strategy in a buffer full situation for the process.
- `atrshmlog_strategy_wait_wait_time`  
You can set the wait time in the wait strategy.
- `atrshmlog_delimiter`  
You can set a new delimiter char in the argv concat of write2.
- `atrshmlog_buffer_infosize`  
You can set the buffer size for dynamic allocated log buffers.
- `atrshmlog_prealloc_buffer_count`  
You can set the number of buffers to allocate in one low level alloc.
- `atrshmlog_f_list_buffer_slave_wait`  
You can set the wait time a slave sleeps if no buffers to work available.
- `atrshmlog_f_list_buffer_slave_count`  
You can set the numbers of slaves to start with.
- `atrshmlog_wait_for_slaves`  
You can set the wait flag for the cleanup for wait for slave to finish first.
- `atrshmlog_clock_id`  
You can set the clock id for get\_clocktime
- `atrshmlog_thread_fence_1 to 13`  
You can set the thread fence flag for the fences.

- atrshmlog\_event\_locks\_max  
You can set the maximum index for events.
- atrshmlog\_logging  
You can set the logging process flag .

There is also a set of values that can be changed by the user with a function.

So this is the first thing. What if it is simply not enough ?

## Changes in the code

Well, as with all open source code – you have it after all on your box right in front of your nose.

So you can make changes.

Its easy. You only have to check for the thing you need to be changes and then make it. Then a makeall.sh and the distribution to the places you need this new thing.

There is one thing you should do at least when you try to change the code – change the version too.

```
#define ATRSHMLOGVERSION (XXX)
```

in atrshmlog\_internal.h.

This is not for making a new module on the internet, its for your local module. You have then a way to circumvent use of mixed modules. If there is any old module in place it will complain about the version. So this is a safety for you. Choose a number of 100 and above, I doubt it will make it with its normal development.

After making a change to the code you can check for the success by using the test programs.

Does it have a positive impact and its OK this way then simply use it.

If you have instead the need that your change has to be in the next version too you have some options.

Making it big, making it small.

Making it big means you have a change for the module that is for the benefit for other users too.

Then you should pack your version, and sent the maintainer (which is me for now) a post card with it in the attachment so he can check for it. If it is a good thing it will be in the next version.

Making it small - if you have only partly this in mind or simply need it only for you you can make it with a patch. Most systems support creating patches with a diff and a patch program. So you create a patch and make it work with the next version by using the patch. No more editing in this case.

This was one reason for splitting up the former one file atrshmlog.c into the impls. Its much easier to handle patches this way.

Still, the version should then be changed. Don't forget this in your patches.

## Adding stuff

OK, you made a new thing now and think its worth the money but you are not selfish so you want to make it into the module.

There are simple questions you should first answer yourself.

- Is it making anything else not to work ?
- Is it making no overhead for others ?
- Is it needing permanent maintenance ?
- Is it of interest of a special group of users ?

Well, that stuff.

If your answers are good, then pack your version and sent me a postcard with it.

For an example we choose the java jni layer. You found a way to log binary an object and that's nice in the converter you patched too.

So your patch consists of a thing that would be very nice to have for all java users of the module.

Does it break other uses ? HM, I guess it will not, so you can answer that with a no.

Will it have overhead for others ? HM, depends. If the new converter has to check for every little entry it could be some seconds in the conversion, but I doubt that would a problem. If it has to fire up a jvm for every single conversion – well, that's different. This would likely end up in a big howling group of C and other users why to have such a thing in there.... OK, for the sake of the discussion its a no, no overhead.

Will it need permanent maintenance ? Well, that's a GOOD question, because you are not the maintainer. So for now if you have that thing in a way the maintainer has to do hours of overtime its bad. For the sake of the discussion it has not.

OK. Is it only of use for a special group ? That's a clear yes. No others than the java developers need such a thing. Or at least need THIS thing. If others need from time to time a similar thing it could be a good start to a plugin feature for the converter – so we simply cannot make it here a no or yes – only if its definitely only for a special group it would be a no.

So now, would it be a disqualification ?

No, because if it is of interest it would be a thing to integrate for the layer in question, and so we would have a new converter for the java folks here in the module.

You see, its not always that clear, but I think in most cases an addition will make it to the layer, or the module. But it depends and so you should give it at least a try.

## **Now that I have it – how do I use it ?**

To use the module should be easy in case you have a C or C++ language environment. At least for the platforms I have encountered its no big problem to link a library or include an object that was made from the system C compiler.

Things can change if you need different compilers, or you are located on the fenster;plural system where you easy end up in the so called dll hell.

But for now we focus on the use in an environment where you have regular support from the compiler and can use the module as is.

## **The way to implement a simple logging program**

This is for the starters, so skip it if you have already done development. We will target a simple C program here.

The module can be used in any C environment by using the headers and the compiled module, the library, or the source itself as include.

So for the simple task to integrate it in a program we start with the well known minimalism C program, known as hello world.

The hello world is doing a lot of stuff, but we will see for that later. For now we start with a simple thing like this:

A screenshot of a terminal window titled "hello\_world.c - emacs@hydra.alphaset.de". The window contains the following C code:

```
#include <stdio.h> // 1
int main(int argc, char*argv[])
{
    printf("hello, world.\n"); // 3
    return 0; // 4
} // 5
```

The status bar at the bottom shows:

U:--- hello\_world.c All L9 (C/l Abbrev)  
Wrote /home/shmlog/docubuild/golden\_1\_0\_0/src/hello\_world.c

*Illustration 23: A hello world program C source code*

Yes, I know its not right, some things are old school and and and ... but it will do the job here.

Compile as C11 and execute it.

Then see for the lines again.

## Include stdio.h

That's the first thing. So we use the whole standard library of the compiler chain here. Most likely you have a glibc in place, or a similar thing. For the UNIX's its different. But you have definitely

not some byte's object here, but a lot of stuff in place you don't know normally.

Make a simple nm of the binary and check what it needs.

```
[shmlog@hydra src]$ gcc -std=gnu11 hello_world.c -o hello_world
[shmlog@hydra src]$ nm hello_world
00000000000509b80 D __TMC_END
00000000000508124 d __tryaltoc_5204
[shmlog@hydra src]$ [shmlog@hydra src]$ nm hello_world
00000000000601034 B __bss_start
00000000000601034 b completed_6940
00000000000601030 B __data_start
00000000000601030 W __data_start
00000000000400470 t __deregister_tm_clones
00000000000400470 t __do_global_dtors_aux
00000000000600e18 t __do_global_dtors_aux_fini_array_entry
000000000004005e8 R __dso_handle
00000000000600e28 d __DYNAMIC
00000000000601034 D __edata
00000000000601038 B __end
000000000004005d4 T __fini
00000000000400510 t __frame_dummy
00000000000600e10 t __frame_dummy_init_array_entry
00000000000400728 t __FRAME_END
00000000000601000 d __GLOBAL_OFFSET_TABLE_
w __gmon_start__
000000000004003e0 T __init
00000000000600e18 t __init_array_end
00000000000600e10 t __init_array_start
000000000004005e0 R __I0_stdin_used
w __ITM_deregisterTMCcloneTable
w __ITM_registerTMCcloneTable
00000000000600e20 d __JCR_END
00000000000600e20 d __JCR_LIST
w __Jv_RegisterClasses
000000000004005d0 T __libc_csu_fini
00000000000400560 T __libc_csu_init
U __libc_start_main@@GLIBC_2.2.5
00000000000400536 T __main
U __putstr@@GLIBC_2.2.5
000000000004004b0 t __register_tm_clones
00000000000400440 T __start
00000000000601038 D __TMC_END__
[shmlog@hydra src]$
```

Illustration 24: A *hello worlds* internal stuff.

## Starting point and parameters

The program will first have to initialize the library, its dynamic memory handling, the environment variables and parameter areas and such stuff. Then the main is hit and your code is on the list. So for now we have to realize there is a lot of initialization before the main starts.

## Calling the library function printf

OK, this is easy. The object already reduced that to a simple puts. Still this means that the library is called, and you can see the `puts@@GLIBC_2.2.5` in place here.

Its not unusual to have the C library on board – at least on a UNIX system – but its possible sometimes to do without it. For the platform I use its still clear that I will have the library on board so I will use it in the module too.

## Returning from main

This is a place were we normally say, well, that's it.

In fact its far from it. There is a lot of deinitialize stuff to do, and we can see this in the nm listing is rough a third of the entry's handling de stuff. So be prepared that a program just not ends. It gives control to the library and then finishes, eventually even a special end stub like the start stup is called, so be prepared for things that can happen after you call things like return or exit.

## **The final end (or not ?)**

After doing all this deinit stuff there is still the cleanup for the process you created. Resources of the system are claimed back from the OS and the OS has to do at least for the file handles and the memory a lot of stuff.

So you see, a hello world is far from the thing you normally are told. Its an opera in several acts.

## **Adding the module**

That's simple. We compiled the module already, so we can use the library. Or make the source include approach. Last thing will be used in the big example below, so I switch to the library here.

First we have to check for the module what has to be done. And we find that we can live for the first test drive with two things here.

First we need the interface of the module. That's the include atrshmlog.h.

OK, the folks of the hard core line will say its sufficient to declare those needed functions simply yourself as extern, but for now I will use the header.

That's the clean solution here.

So we need the include of it in place.

The screenshot shows an Emacs window titled "hello\_world.c - emacs@hydra.alphaset.de". The menu bar includes File, Edit, Options, Buffers, Tools, C, and Help. The toolbar below has icons for New, Open, Save, Undo, Cut, Copy, Paste, and Search. The main buffer contains the following C code:

```
#include <stdio.h> // 1
#include "atrshmlog.h" // 6
int main(int argc, char*argv[])
{
    printf("hello, world.\n"); // 3
    return 0; // 4
} // 5
```

The status bar at the bottom shows "U:--- hello\_world.c All L4 (C/l Abbrev)" and "Wrote /home/shmlog/docubuild/golden\_1\_0\_0/src/hello\_world.c".

*Illustration 25: The include added*

Its no special thing to include it, so you can do it before or after the stdio.h, itself does not include that thing. In this version there is only one include that is done by the atrshmlog.h itself, and that's the stdint.h. So we add this here and have no change for the program else.

So far we have now the most functions available, we have to call the first vital one. Its the atrshmlog\_attach().

## Attaching to an area

We have to connect to an area. No area, no logging. Period.

So this is done with the attach function.

It takes its knowledge from environment variables or flag files, so there are no parameters here. We simply have to call it.

Why we ?

OK. This is perhaps worth the discussion .

Its theoretical possible to do the initialization together with the first use of the log.

That's called lazy evaluation in many programming community's.

I decided not to do this.

Lazy evaluation would shift the point of error for the log to a first use situation. And that's more than bad if you think in terms of separation of responsibility's. The attach and the possible error situation should be clear and so the log simply should do an explicit attach and initialization operation.

This has the advantage that you have control over the things done. And you can be sure that the code you write can make the decision what to do. And not some deep down buried mechanism inside a who knows when called piece of code.

On the bad side this means that its possible to make some logging before the program executes the attach. So the module must handle the infamous problem of use before initialization. That's a problem that becomes vital in C++ program, where you can use initialization code for your statics. In C its less a problem. So for now we ignore that.

We add the attach.

The screenshot shows an Emacs window titled "hello\_world.c - emacs@hydra.alphaset.de". The buffer contains the following C code:

```
#include <stdio.h> // 1
#include "atrshmlog.h" // 6
int main(int argc, char*argv[])
{
    intn attach_result = ATRSHMLOG_ATTACH(); // 7
    printf("hello, world.\n"); // 3
    return 0; // 4
} // 5
```

The status bar at the bottom of the window displays the file name "U:--- hello\_world.c All L7 (C/l Abbrev)" and the message "Wrote /home/shmlog/docubuild/golden\_1\_0\_0/src/hello\_world.c".

*Illustration 26: Attaching to the area.*

Now we can add the one logging call we need.

## **Adding the logging**

So all we need now is the one log call we want to make here.

If we ignore the result of the attach it will end up in checking an event that's not set and an global process logging flag that's 0. So no damage is done. Only some cycles of your CPU will be needed to realize that the log cannot be made.

That's the theory. In practice you have to realize that there can be hidden costs if you play games with things that are needed for the logging, especially when it comes to building log info.

So you should ask yourself if you can live with that too.

If not, then try to make use of the return value, or use the Macro ATRSHMLOG\_EVENTCHECK() here.

For our small example we have only a static text, so it is no problem to do it simple here.

The screenshot shows an Emacs window with a dark theme. The title bar reads "hello\_world.c - emacs@hydra.alphaset.de". The menu bar includes "File", "Edit", "Options", "Buffers", "Tools", "C", and "Help". Below the menu is a toolbar with icons for file operations like Open, Save, Undo, and Search. The main buffer contains the following C code:

```
#include <stdio.h> // 1
#include "atrshmlog.h" // 6
int main(int argc, char*argv[])
{
    int attach_result = ATRSHMLOG_ATTACH(); // 7
    printf("hello, world.\n"); // 3
    ATRSHMLOG_WRITE(1,
                    'P',
                    1,
                    0,0,
                    "hello world",
                    sizeof("hello world") -1
    ); // 8
    return 0; // 4
} // 5
```

The status bar at the bottom shows "U:--- hello\_world.c All L8 (C/l Abbrev)".

*Illustration 27: Adding the logging ( and correcting a nasty error too)*

OK. Now after we have it we compile it, and link the thing like the programs of the module.

## Compile and test

So its after all the object liked together.

```

src : bash — Konsole
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
0000000000400470 t deregister_tm_clones
00000000004004f0 t __db_global_dtors_aux
0000000000600e18 t __do_global_dtors_aux_fini_array_entry
0000000000400588 R __dso_handle
0000000000600e28 d DYNAMIC
0000000000601034 D __edata
0000000000601098 B __end
00000000004005d4 T __fini
0000000000400510 t frame_dummy
0000000000600110 t __frame_dummy_init_array_entry
0000000000400728 r __FRAME_END
0000000000601000 d __GLOBAL_OFFSET_TABLE__
w __gmon_start__
00000000004003e0 T __init
0000000000600e18 t __init_array_end
0000000000600e10 t __init_array_start
00000000004005e0 R __IO_stdin_used
w __ITM_deregisterTMCloneTable
w __ITM_registerTMCloneTable
0000000000600e20 d __JCR_END__
0000000000600e20 d __JCR_LIST__
w __Jv_RegisterClasses
00000000004005d0 T __libc_csu_fini
0000000000400560 T __libc_csu_init
U __libc_start_main@@GLIBC_2.2.5
0000000000400536 T main
U puts@@GLIBC_2.2.5
00000000004004b0 t register_tm_clones
0000000000400440 T __start
0000000000601038 D __TMC_END__
[shnlog@hydra src]$ g99 hello_world.c
hello_world.c: In Funktion `main':
hello_world.c:7:3: Fehler: unbekannter Typname: »itn«
  itn attach_result = ATRSHMLOG_ATTACH(); // 7
^
[shnlog@hydra src]$ g99 hello_world.c
[shnlog@hydra src]$ ell hello_world
[shnlog@hydra src]$ ./hello_world
hello, world.
[shnlog@hydra src]$ ■

```

*Illustration 28: The build and a first test.*

Great. At least it seems it does not crash (core dump is possible for the account I use) and that's it.

To see the log we need to make the area, initialize it, use the thing, take the reader to get the output and convert to text.

That was already part in How to build chapter, so see there. Here is the output when its done.

```

src : bash — Konsole
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
ATRSHMLOG_ID="10649633"
export ATRSHMLOG_ID
if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
[shnlog@hydra src]$ ATRSHMLOG_ID="10649633"
[shnlog@hydra src]$ export ATRSHMLOG_ID
[shnlog@hydra src]$
[shnlog@hydra src]$ atrshmloginit 8
shm log attach and init.
logsystem version is 1
[shnlog@hydra src]$ ./hello_world
hello, world.
[shnlog@hydra src]$ atrshmlogoff
shm log attach and set ich_habe_fertig to 1.
logsystem version is 1.
ich_habe_fertig was before 0
[shnlog@hydra src]$ atrshmlogrederd d2
bash: atrshmlogrederd: Befehl nicht gefunden...
[shnlog@hydra src]$ atrshmlogreaderd d2
shm log attach and loop write file.
logsystem version is 1.
directory is d2
files_per_dir is 10000
directory not found or not accessible 'd2'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
logging done.
writer data : time 210070 count 1 per use 210070
[shnlog@hydra src]$ atrshmlogconv d2
shm log converter from file 'd2/0/atrshmlog_p21784_t21784_fo.bin' to file 'd2/0/atrshmlog_p21784_t21784_fo.txt'.
id 1 acquiretime 880 pid 21784 tid 21784 slavetime 580 readertime 1470 payloadsize 40 shmbuffer 0 filenumber 0
[shnlog@hydra src]$ cat d2/0/atrshmlog_p21784_t21784_fo.txt
[shnlog@hydra src]$ cat d2/0/atrshmlog_p21784_t21784_fo.txt
0000021784 0000000000021784 000 000000000000000000000000 000055167672793470 000055167672793470 00000000000000000000000000000000 1476369624677945051 1476369624677945051 00000000000000000000000000000000
[shnlog@hydra src]$ ■

```

*Illustration 29: The output after a test against an active area (and some ups)*

OK,for an old man doing it by hand its OK.

So....

## The deep stuff

.... now to the line 8.

See it again:

```
ATRSHMLLOG_WRITE(1,
    'P',
    1,
    0,0,
    "hello world",
    sizeof("hello world") -1
); // 8
```

*Illustration 30: The deep stuff*

OK. There is now a lot to say here about how to make a log entry.

Please check after this the HTML or man or latex documentation. Try to become familiar with the way such things are said in the online help. This makes it easy for you to see for the other stuff what's perhaps really in there.

First of all we have here again a MACRO.

I didn't say this before, but using the MACRO of a function is the preferred way if you use the module.

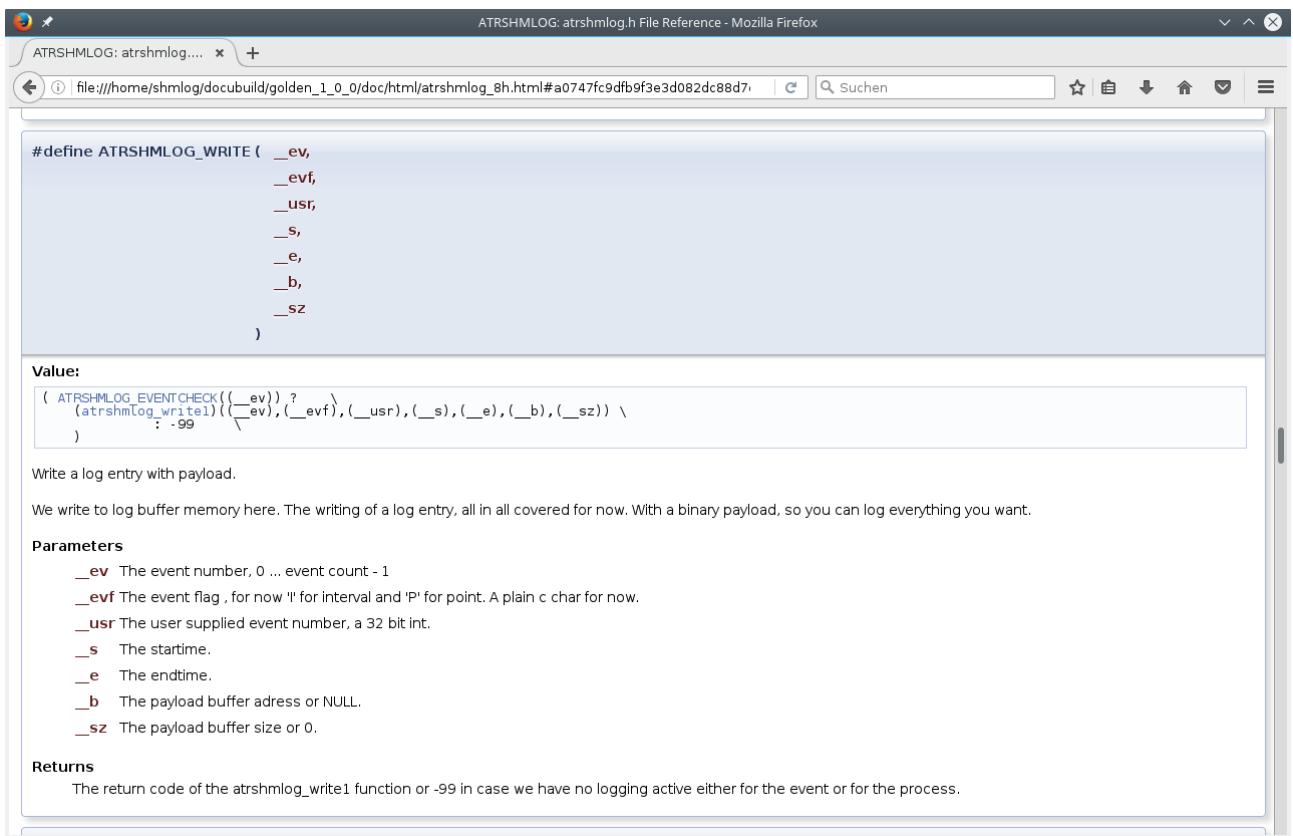
There is some kind of hidden test here in place, so it's definitely better than the call of the function atrshmllog\_write1(). And as always when it comes to a C library it's a kind of last defense line against last minute changes in the module itself. You only use a macro and the guy in the backyard can make adjustments you don't have to know.

In this case we have the eventcheck right before in place, so the function will not be called if that check gives a no go for the event number 1.

Next there is the set of parameters, all seems simple for now. A number, a character, a number, two zeros, a string constant – OK, that's a bit wired, a sizeof of the thing and a – 1 .

First things first.

Check now the HTML documentation for the Macro:



*Illustration 31: The online documentation for ATRSHMLOG\_WRITE*

OK. The first number is the event. So we use that event in case we want to switch that thing on or off. Per default the events are on when we are attached, and off when we didn't make the attach. The attach can make some tricky things in the initialization, so see for it when you have time for this. You can set events on, off and define the meaning upside down to minimize your work for the things.

For now we know we can switch the logging off if we set the event 1 to off.

Next is the eventflag, that is a character and it is needed in the transfer chain at several points. For now simply accept that we take a big P and mean that we have a point in time log entry. And its a C string type info.

Next is the user event. Its an info we can use, so I used her a 1. If I had an info worth it and fitting in an int32 value I could give it here. Often this is the case when a loop is in place. So you get the picture. Use YOUR favorite thing here. The event is strictly for the module used, and the eventflag too, but for the userevent its your free thing to use.

Now comes the pair of 0 values, and we see that this should have been the starttime and endtime.

OK, here comes the big info for you. If you do a log and you do a point in time type, the endtime is always set to the starttime. So the second 0 is simply there to make the compiler happy.

You can of course now start to write a bunch of different macros to cover this. But you will not gain additional functionality, so for now I have at least accepted that I give it then a zero.

For the starttime its different. If you have one you can put it in, if not a zero here means the thing will call internal the gettimeofday function and use this. And because of the point in time thing its the same value that ends up for the endtime.

OK. This is a little bit tricky, but it saves really time and space in many cases. For the not point in time its the endtime that is fetched internal, so you need only the starttime there.

And for the layers its a big difference to need a gettimeofday call or not. So this is a hidden mechanism, but this is a thing I had to accept after the tests and so I broke here my rule of separation of responsibility's – you can get the time not only with the gettimeofday function, but also in a deep buried way in the write functions too.

The next is an address of a buffer, and the last the size of the buffer.

Surprise.

The write uses not the natural way to transfer a C string – this would be easy to do, and it would mean for many uses to do it right. But instead we use a buffer with start and length.

So we do a binary transfer here, and yes, we can do that for real binary data.

Its up to the convert to handle the things we log. The transfer is itself always binary, so we don't see any changes for the payload of the log call till it ends in the file written by the reader.

This has a big impact: We have to know the start and size of the thing, so we call in doubt a strlen or have a calculated result at hand for the size (that's the sizeof here).

Its then our program that crashes if we had the much to often happen bad luck thing with C strings.

Not the module. You will see this in the debugger in case it happens.

On the bright side we can log what we want, from strings to structures to raw memory. As long as we know what we do it will be transferred. If it is not too big, of course.

The bad side of this is that we have in doubt to change the converter. But this is another story. See the chapter for the converter for this.

OK. Now that we have realized the way the write transfers its clear what it is all about the last parameter – we let the compiler calculate the buffer length, and because this will involve the terminating 0 byte we get rid of it by reducing the size by one.

And now back to the log output.

Here is the line we got from the converter. This time we name the fields one by one

- 0000021784

That's the first number, and it is the process id. Check for it when you run the program.

- 0000000000021784

That's the thread id. Its the same for such a simple program as hello world. For programs that use multi threading its different. On my platform I have access to the Linux thread id, so I use this here, on others it could be also a different thing like a pthread id or worse. But its for now a 64 bit number in the system.

- 000

This is the buffer id that was given to the buffer in the program. We can see this way when we switch buffers. And when we recycle them for different threads.

- 000000000000000000000000

This is the file number that was used by the reader to write the log down. This can be used to see if we have a change in the use of buffers in the file system.

- 000055167672793470

This is the starttime. Its a clicktime, meaning we have here the clicks of the tick counter register of the CPU – not a real time, but the best thing a CPU can give you to know when a thing happens.

- 000055167672793470

The endtime. As already mentioned in this example a copy of the starttime. Next example will use a different one.

- 000000000000000000000000

The delta for starttime and endtime. In this example its zero ( Why ? Can you figure it out ?).

- 1476369624677945051

The estimated real time for the starttime. The converter uses two mealtime measurements and calculated an estimation for the time that corresponds to the click time. Its the number of nanoseconds since the system starts to count. In case of my platform its the time since 01.01.1970 00:00:00: UTC. In nanos.

- 1476369624677945051

The endtime.

- 000000000000000000000000

The delta.

- 0000000001

The event of the log entry.

- P  
The eventflag of the log entry.
- 0000000001  
The userevent.
- hello world  
The payload.

The converter has some stuff inside to make multiple lines if you insist to have them in the payload.

So for our first try its OK.

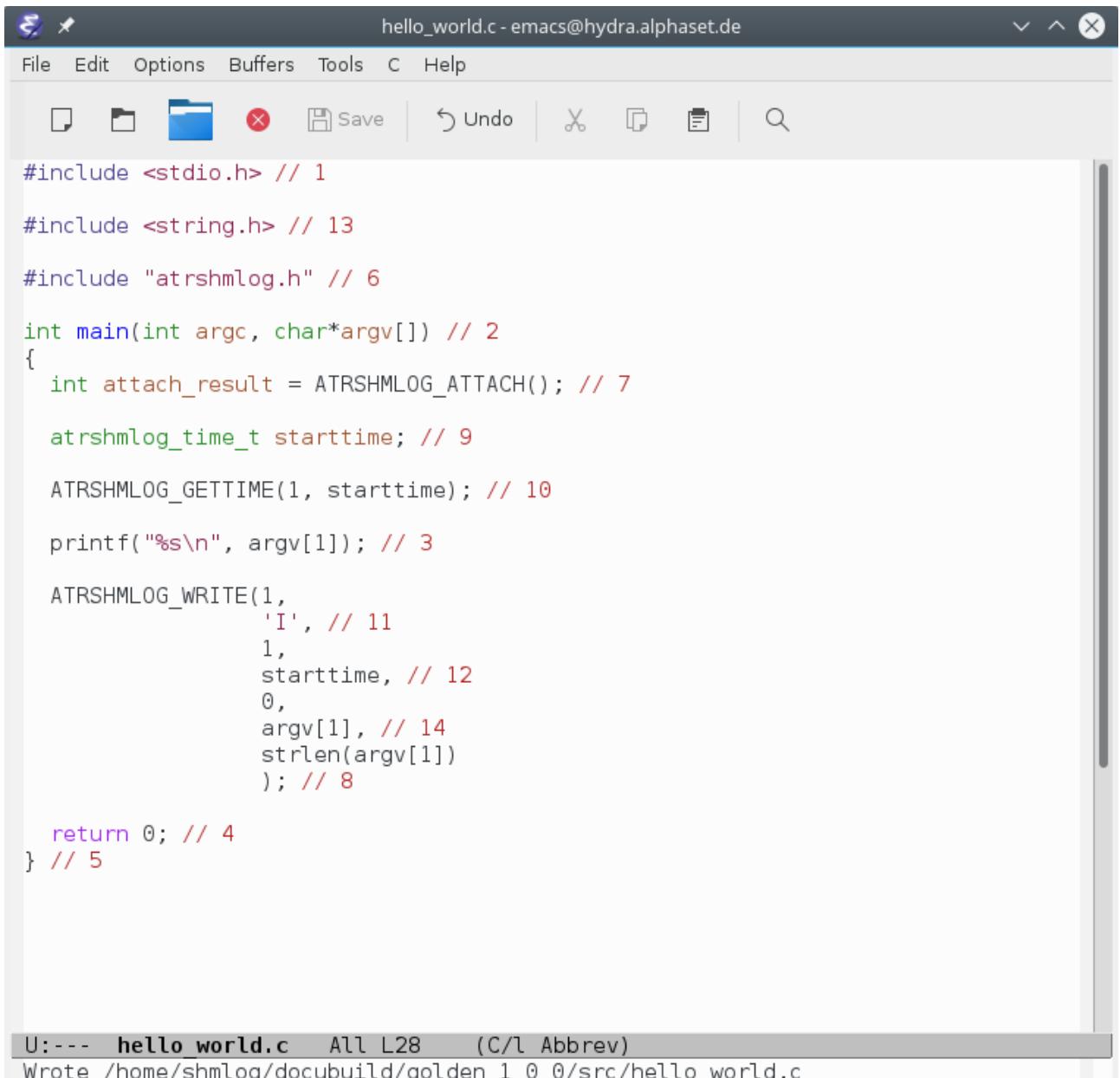
Now we have done this we can start to do some interesting things.

We build in a gettime for the start, a endtime for the printf and then see what that costs.

And to make things interesting we use this time the arguments of the program.

## Measuring a printf

OK. That's simple. We add a gettimeofday for the time and then do an interval instead of a point in time log.



The screenshot shows an Emacs window with the title "hello\_world.c - emacs@hydra.alphaset.de". The menu bar includes File, Edit, Options, Buffers, Tools, C, and Help. The toolbar contains icons for file operations like Open, Save, Undo, and Search. The main buffer contains the following C code:

```
#include <stdio.h> // 1
#include <string.h> // 13
#include "atrshmlog.h" // 6

int main(int argc, char*argv[])
{
    int attach_result = ATRSHMLOG_ATTACH(); // 7

    atrshmlog_time_t starttime; // 9

    ATRSHMLOG_GETTIME(1, starttime); // 10

    printf("%s\n", argv[1]); // 3

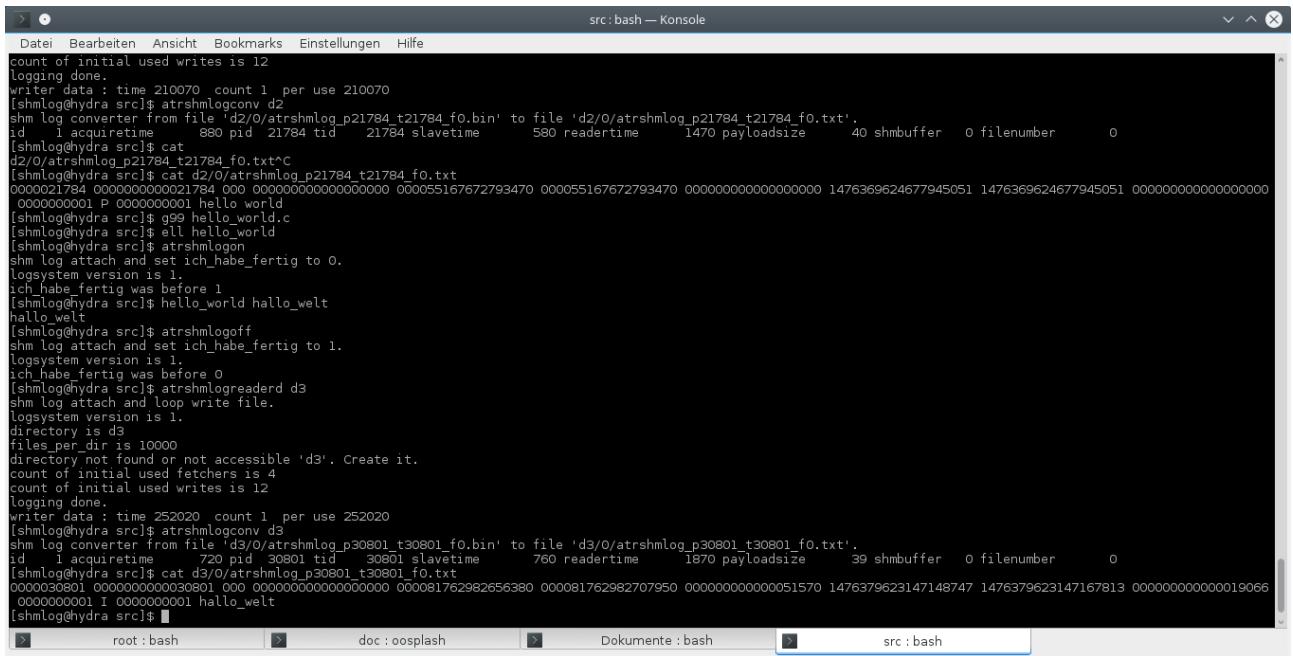
    ATRSHMLOG_WRITE(1,
                    'I', // 11
                    1,
                    starttime, // 12
                    0,
                    argv[1], // 14
                    strlen(argv[1])
                    ); // 8

    return 0; // 4
} // 5
```

The status bar at the bottom shows "U:--- hello\_world.c All L28 (C/l Abbrev)" and "Wrote /home/shmlog/docubuild/golden\_1\_0\_0/src/hello\_world.c".

Illustration 32: Hello world with use of argv and timing the printf

That made it then to this



The screenshot shows a terminal window titled "src : bash — Konsole". The terminal displays a series of commands and their execution times. The user runs "attrshmconv d2" to convert a log file. Then, they run "attrshmlogon" and "attrshmlogoff" multiple times, each taking approximately 21 seconds. They also run "attrshmlogreader d3" once, which takes about 25 seconds. The user then runs "cat d3/0/attrshmlog\_p30801\_t30801\_fo.txt" to read the log file, which takes about 7 seconds. The terminal also shows various log messages related to file operations and system logs.

```
count of initial used writes is 12
logging done.
writer data : time 210070 count 1 per use 210070
[shmlog@hydra src]$ attrshmconv d2
shm log converter from file 'd2/0/attrshmlog_p21784_t21784_fo.bin' to file 'd2/0/attrshmlog_p21784_t21784_fo.txt'.
id 1 acquiretime 880 pid 21784 tid 21784 slavetime 580 readertime 1470 payloadsize 40 shmbuffer 0 filenumber 0
[shmlog@hydra src]$ cat d2/0/attrshmlog_p21784_t21784_fo.txt
[shmlog@hydra src]$ cat d2/0/attrshmlog_p21784_t21784_fo.txt
0000021784 0000000000021784_000 0000000000000000000055167672793470 000055167672793470 00000000000000000000 1476369624677945051 1476369624677945051 00000000000000000000
000000000001 P 000000000001 hello_world
[shmlog@hydra src]$ g99 hello_world
[shmlog@hydra src]$ ell hello_world
[shmlog@hydra src]$ attrshmlogon
shm log attach and set ich_habe_fertig to 0.
logsystem version is 1.
ich_habe_fertig was before 1
[shmlog@hydra src]$ hello_world hallo_welt
hallo_welt
[shmlog@hydra src]$ attrshmlogoff
shm log attach and set ich_habe_fertig to 1.
logsystem version is 1.
ich_habe_fertig was before 0
[shmlog@hydra src]$ attrshmlogreader d3
shm log attach and loop write file.
logsystem version is 1.
directory is d3
files_per_dir is 10000
directory not found or not accessible 'd3'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
logging done.
writer data : time 252020 count 1 per use 252020
[shmlog@hydra src]$ attrshmconv d3
shm log converter from file 'd3/0/attrshmlog_p30801_t30801_fo.bin' to file 'd3/0/attrshmlog_p30801_t30801_fo.txt'.
id 1 acquiretime 720 pid 30801 tid 30801 slavetime 760 readertime 1870 payloadsize 39 shmbuffer 0 filenumber 0
[shmlog@hydra src]$ cat d3/0/attrshmlog_p30801_t30801_fo.txt
0000030801 0000000000030801_000 0000000000000000000000000000081762982707950 000000000000000051570 1476379623147148747 1476379623147167813 000000000000000019066
000000000001 I 000000000001 hallo_welt
[shmlog@hydra src]$
```

Illustration 33: The test for hello world with argv use and printf timing

Wow. 51000 clicks for the printf thing.

Now we can make tests with it – I leave that to you. Try at least to test the time needed for the gettimeofday and the write with a static string. Then make some tests for the printf and the file operations fopen, fwrite, fflush etc.

After this its time to switch to a big example.

## The big example for the C community

OK. Now its time for something greater. We use an exiting program source code here and implement the logging.

The thing was simple. Support the korn shell.

This said I made the first log version and checked for the korn shell code – I knew the guys at former bell labs were today doing things in a relative open way, so the source code should be there. And as a hint I had already seen a ksh – not the pdksh – on a system about three years ago....

I found a dead end after the other. Yes, there was the source and yes, that server was down. The usual thing when a guy leaves the company and the company starts to forget....

Try to keep it up, but ...

... did made it again run, Try again and ...

... down again....

BAD.

So I switched to the distro at hand. Found a rpm. Found the source rpm.

Surprise.

The thing used an own build system. And it was not there. Only the source tree for the shell.

Switch to another distro. Check for ksh. Bingo. Found rpm. Found source rpm.

This time luckily with the build environment ( Its the next generation make of those guys, called nmake – which was a good idea of the guys, but a little software company at Redmond had the same idea for a bastard that I nether liked at all .... so all hits for nmake were a real punch in the air).

OK. Adapting to my distro.

OK. Building the stuff.

OK. Checking for the compiler I used (was c99 that time, not c11, but worked).

OK. Find the main and then see what you can do.

OK. Include done, module added as an include (was one file at that time).

OK. Thing started, connected to the area and made logging.

OK. Try to understand how the source worked, try to implement useful logging.

OK. Had it done after 5 days ( 5 ? wow, didn't thought it would be possible, but OK).

When I then came into contact for the technical guy – Mr. Sven Winnecke - ....

Surprise.

Its not the korn shell of David Korn, but the mksh of the MirOS BSD.

Umpf.

Check for the source. Got it.

Bring it to run on the distro.

OK. I could build.

OK. Check for main.

OK. Attach is done.

OK. Logging works as before.

OK. Try to implement useful logging.

OK. Had it done after 5 days (again ... only 5 ?).

The next months were a permanent refinement of the logging itself, and from time to time changes in the source to make a better logging. Most things in the source had to do with the problem that there was no clear info in the thing about the most wanted info in a log – the position in the system – this time not the position of the writes, that was easy, but for the executed intermediate code the line in the corresponding shell script. This was about 80 % of the work and has nothing to do with the log itself, its the simple problem in case you don't have the thumb on it in the phase of design and evolution to make a thing work that any user suspects to be there out of the box.

OK. After all that fighting with source code of other people – it worked.

I got the line number with an approximation in the case it was not there and carefully changed the code so it was there where I needed it.

After two months I could focus on the log itself again.

Making it faster, making it better for multithreaded programs and all the stuff for mutex's, then condition variables, support threads and at last – atomics. And the memory model thing ....

Back to the thing at hand.

Its the mksh version 52 b. Now replaced by c, and I have no intention to do it again. So 52 b will be our big example.

## main

You find the code in BASEDIR/mksh/mksh.logging.

First thing is of course main. You open up your favorite text editor and then in main.c it is.

You have the unchanged source in the tree BASEDIR/mksh/mksh.52b for reference.

Remember the hello world ?

Adding the header, adding the attach.

See line 819 and follows. I use some time variables here. Needless to say I did some changes for the type, but it was always that simple use of variables here.

Just to find about some speed timings I made some otherwise not useful logging's. Simply ignore them.

After the attach in line 831 we can skip to 990. The thing uses an initialization function here, so making some timings here is a real thing to do.

Ever had discussions about program start up ? Well, that's now a thing you can check for...

I use here a simple approach for the events numbering. Every implementation file gets its own hundreds number range. So I use here 1 to 99. And after playing some games I came up at end with using the same event for the time measurement call and the log call. Oh, and I didn't had the hidden timing in the write at that time, so I do every gettimeofday for endtime, too.

In line 1007 is one of the vital branches. The main interpreter loop. In 1022 the other loop.

OK. This means some playing with if else and making new events to make possible to switch off timing and logging I don't need.

Back to the initialization, go to line 181. main\_init.

Timing is OK. Make a gettimeofday. Then long time no need to do a thing – its after all internal stuff of the shell, so I don't need times here. The interesting spot is at 370, where I have to give the shell the knowledge of the variables I used in the attach – or better – the name and value it should use for it.

Making a shell logging is one thing. Having it running in login shell mode is another. You cannot simply set system wide variables. So you need something different. In this case it were the flag files.

So now you know why they here. Its for a logging shell and you cannot set environment variables before it has started.

And yes, I could live with a “so what ?” here – but I wanted the values most vital after the start inside as variables. So I had to set them in the shell's init . Here.

Next thing interesting was the file include stuff. Every shell starts in normal mode and reads in include files.

A profile here, some alias file there, eventually. A history ...

So from line 524 on we have the include stuff and its timing.

Ever had discussions about the time you need for start up and config files ? Now we know that....

Things are a bit strange to me, but in line 670 we are back on the money for init file timings.

I line 800 we have done so far that init stuff.

There is one dirty thing here. The line number problem. Solution was a more or less heuristically one : make a line number guess and keep it up to date when you have one. So I introduced in the – single threaded – shell source a global variable, g\_linehint. Every time I have one I set it. When I don't have one, I use the g\_linehint.

That's a bit dirty, but the result is to have a line number that is most accurate for the logging, only one line off some time. I can live with that.

Include is the next interesting thing. You take a file and include it. Again is the timing of interest, and the logging needs a good line number – of the processed file of course, not the file the include is in, but the file you process now...

We start in 1056 and take our time. The some vital info's – which include.

Ever have tried to figure out how many files you process in a shell start up on a system – and what files ? Now we know that.

Next thing is the command. Most wanted for timing. How long took ...

Turns out its the next thing, shell , we need for it.

Its too much to give back in a simple text, so I spare you most of it. The case is perhaps the only thing you should investigate.

The rest is support stuff, perhaps someone needs logging here, but not for me...

At the end in line 2556 I did the one and only thing I needed at that time – include the module itself as source. Today it is needed to do this with the library, so this is now wrong.

## **eval**

Its more than interesting to see how the shell handles a diametrical conflict – execute scripts as fast as you can on one side, execute dynamic stuff on the other side, and last doing interactive input handling. So eval is a vital spot – you call it more often than you think.

OK. include made in 26.

Ups, what's that ? Substitute ? Interesting. Give it a timing and log.

Next is evalstr, the thing is used at several places. Also a vital target for logging. BAD THING: more code for error handling and logging in those cases than real code. Sometimes logging can be elegant, but that low level thing is not.

The expand then brings lots of parsing stuff. So logging and timing again. OK. We are already over 50 % ....

Then only supporting stuff. No need for more logging.

## **exec**

Most likely the most important thing – the executor. The shell uses a scan compile execute internal code approach. So this is the thing that really does the things we want to do.

Right in line 75, we start our logging's.... Good new: we have – sometimes – line numbers and can update our heuristic helper hack.

Bad news, only part of the time ...

Well, this sucks, but I can't change it. So some playing with variables here. Then back to business. Line 205.

THE MAIN INTERPRETERS SWITCH.

Bingo.

From here on we can tell what a simple command takes – i.e. a program call.

And ( command ) sequences, and | ing, and the not so often seen ; things ....

And also some less often seen things, like the co process.

All in all this is the real jackpot, till we reach line 800. Simply get the info you have and write it out.

Some helpers need attention too. Comexec is one of them. After some iterations we have a logging for the external binary's. Wow.

Bad thing. Need another helper variable, the command name . But its local, so I guess it hurts not too much. Again we get sometimes line numbers ...

define. Nice . How long takes it to make a variable ... or function... or undefine it ....

And again a nice one. Built in.

Ever had discussions about using internals or external programs ? Or functions ? Now we know that.

Skip some support stuff. No need for a log. Hit 1934. The builtin execution.

There are two bastards, one is WAIT and one is SLEEP.

Here we are for the wait.

Last action in this file is iosetup. Ever had discussions about the speed of the io operations for the shell ? Now we know that.

## **expr**

This is also a vital thing – anything you try to do in the right side of ....

Starts with line 205, v\_evaluate.

But I overdid it. So after some iterations its the only place now.

## **funcs**

OK. Here are the next targets. We start with the smallest of all, the . Command.

Line 1800 and we are in. Sourcing of files in a already compiled script context – how often did we discuss if it were too expensive to make include files to source ... Now we know it.

Line 1910 , the WAIT.

The rest is support stuff, no logging needed.

## **The rest**

Well, we could use eventually. Some additional logging, but for my purpose its enough.

Build the thing, set up a working area, run and see for yourself.

Never again discussions about times for functions, scripts, programs ...

How nice that could have been.

Sad to say, but they never needed such a thing.

So this is now the one big example how to log ( and its a boring one, only the usual stuff, and no real new functionality ) - but its there and if you are interested you can make it run – remove the source include in main, link a library and you are in.

Needless to say that I had two times changed the use of the macros, so I had to work it again two times, but after all it was relative stable.

## The java language support

The java support was done after the development of the mksh logging. So in this phase the need for multi thread logging entered the scene. Before that it was – well, a simple thing to log in a simple big system, the mksh and former the ksh. From now on the thing changed dramatically.

There was a simple problem.

Logging was done with real-time time stamps – get\_clocktime on Linux – and worked in an approach to direct transfer to shared memory buffer.

The tests with the thing made it clear that this could not work in a production environment like an application server.

So it became a change for the timing model – the click time approach – and an even more drastic change for the logging itself. It became a thread oriented logging with alternatively used buffers.

So the java support was done with that in mind.

When the time was right, a first approach to a jni base support layer was done.

After some iterations the layer is now in place, and you have near full support for the module.

You can use it even without the need for the C programs if you want – every function of the C module is supported on the java layer's jni module.

In practice its better to do things not twice, so I abandon the plan to deliver also java support programs in this version. That's up to the java community for now. But its possible to do at least the needed stuff for making areas and destroying them, so your work flow for testing is save. And if you made it to support programs I will happily integrate them here. You can use as a starting point the tests subpackage code...

## How it works

The module is a jni implementation. Under cover you use the C module. The module is capable of doing its job simply by using the internal infrastructure as in a C environment. No need for changes inside.

This means you get in theory the same speed as for the C module itself. The only thing that slows down is the jni bridge itself.

So when it comes to timing its near 50 % of the raw C function. For logging its a bit slower, and because of the needed copy thing the module switched after some testing to the so called critical methods of the jni interface. No copy of string or arrays if it is not needed, only use as a read only address of data. Still for portability a use of the UTF converter if you need it.

The rest is more or less simple. Use of the function and no object stuff. Simple raw data types, no classes, no exceptions and no use of the java side. Barely some string creations.

If you are interested the atrshmlogjnipay.c is a good start.

## How to use it

The user needs the following parts:

- Class ATRSHMLOG.java

This is the low level jni layer class. And its the only class you will get for now for use. Check the implementation and focus on the backbone methods first – attach, gettime, write. The rest is simply there to make the module fully accessible and in case of need you can make use of create, delete, read and all other things.

- Shared library or dll atrshmlogjni

This is the bridge code from atrshmlogjnipay.c and the linked library from the C module, libatrshmlog.a. You need also the compiler support lib's and in case of the mingw port the additional dll 's of the mingw system.

- The support programs

You need the matching C support programs for the basic stuff you don't want to reinvent. Reader, converter, even create and delete. So if you need a pure (?) jni only solution you can replace them with your own code java counterparts. But for now I deliver nothing so you should start with the C programs.

The rest is to copy the things created in the build process into something javaish – a zip or war or whatever thing – and then use it. The raw class will do fine for the start, but don't forget at least the other class files.

So you first have to build it – again I am not delivering binary code, only source, so if you are interested in binary you have to contact me and I will see what I can do. But for now you have to start with ATRSHMLOG.java and the jni bridge file atrshmlogjnipay.c .

## How to build it in the first place

We start with the C module.

After this is in place we have to check our next options.

We have in place implementations that should work out of the box for oracle jdk, openjdk and IBM jdk.

The oracle jdk is available for Linux – this was the major development platform. Its an outdated jdk 1.8.0 66 , but for jni its nearly identical for version 1.5 up to now.

The oracle jdk was also used for the mingw cross compile port for a win 7 system. So if you need this its best to start with the jdk 1.8.0 102. As for Linux it should work nearly same from java 1.5 on to now.

The openjdk is more or less the backbone today for the oracle jdk, so no surprise that this works

too. It was first done for fedora and 1.8.0 111, but it should work for others identical.

The IBM jdk was made last and for Linux again first. Its a new 8.0 version. Should do it for fenster;plural too, but didn't test that. Should work from version 5.0 on to now.

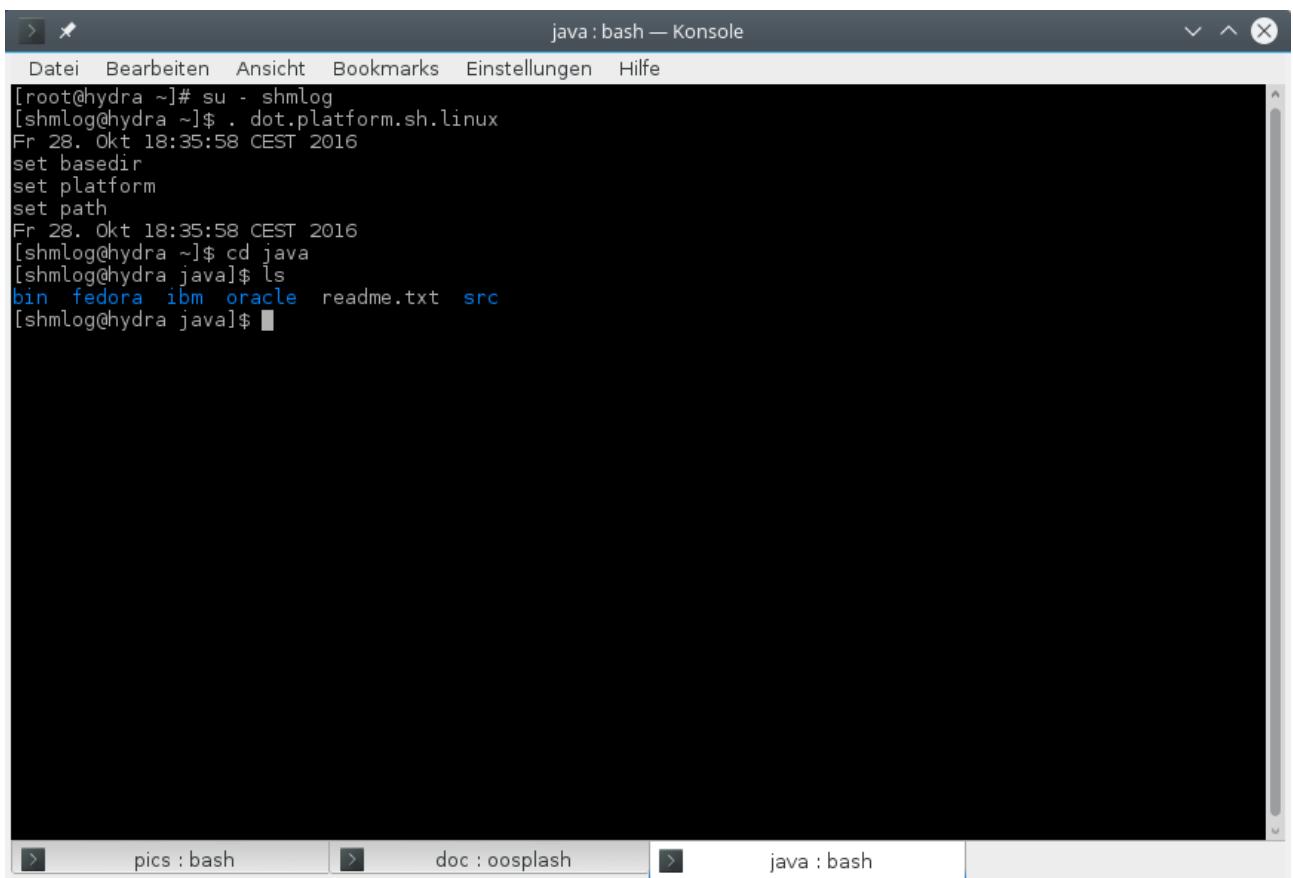
So you have four jdk at hand that made it so far.

You know what you want to do – take one of these, check for your needs and then start working.

## ***The java directory***

We start with the basics.

The java stuff is located parallel to src – so we start at BASEDIR/java.



```
java : bash — Konsole
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
[root@hydra ~]# su - shmllog
[shmllog@hydra ~]$ . dot.platform.sh.linux
Fr 28. Okt 18:35:58 CEST 2016
set basedir
set platform
set path
Fr 28. Okt 18:35:58 CEST 2016
[shmllog@hydra ~]$ cd java
[shmllog@hydra java]$ ls
bin fedora ibm oracle readme.txt src
[shmllog@hydra java]$
```

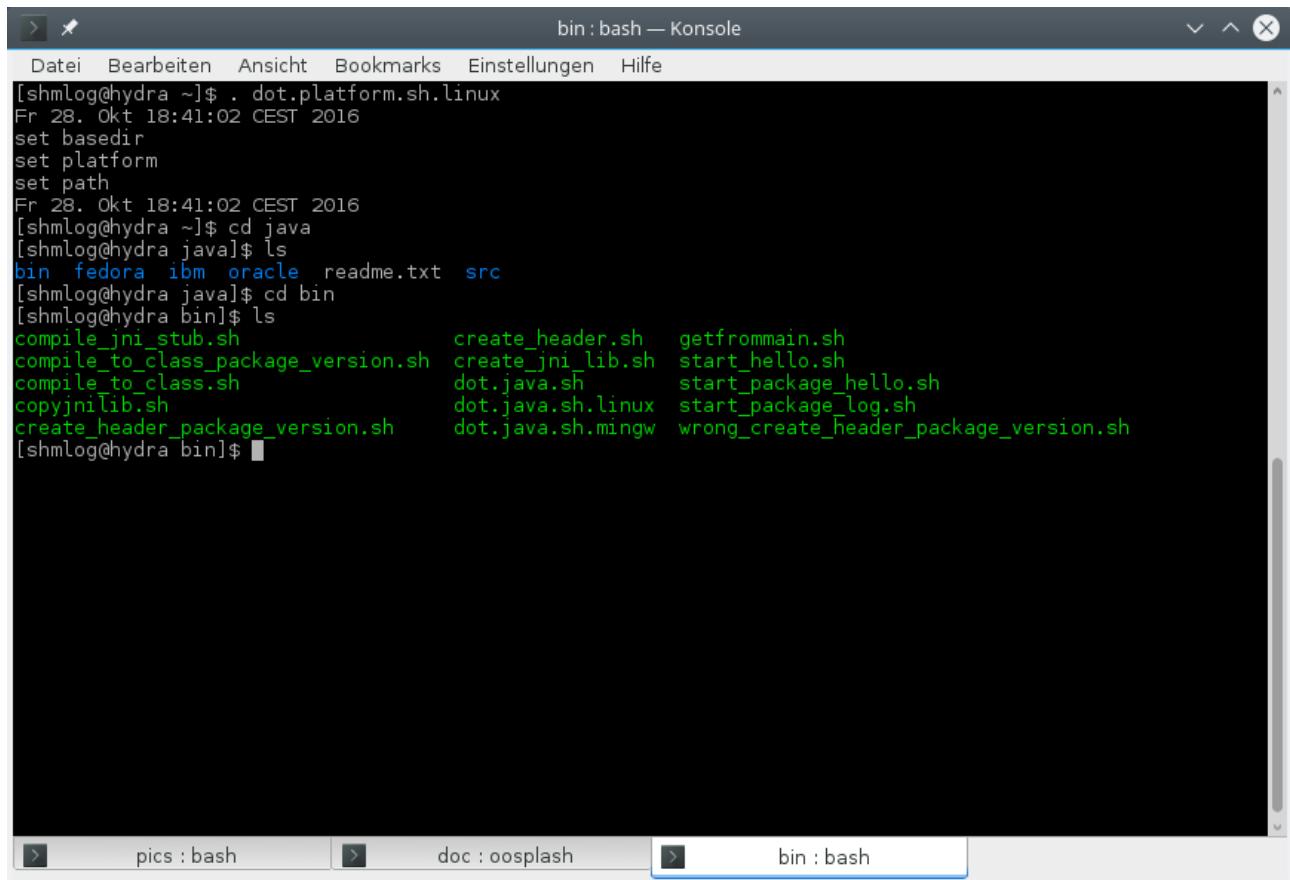
*Illustration 34: The java ase directory*

OK. We have a readme – check it – and a bin and a src. And then oracle , IBM and fedora.

For the bin its clear, there are some scripts.

## The bin directory

That's my bin so far.



A screenshot of a terminal window titled "bin : bash — Konsole". The window shows a file tree with several scripts in the bin directory. The scripts include: compile\_jni\_stub.sh, compile\_to\_class\_package\_version.sh, compile\_to\_class.sh, copyjnilib.sh, create\_header\_package\_version.sh, create\_header.sh, create\_jni\_lib.sh, dot.java.sh, dot.java.sh.linux, dot.java.sh.mingw, getfrommain.sh, start\_hello.sh, start\_package\_hello.sh, start\_package\_log.sh, and wrong\_create\_header\_package\_version.sh. The terminal also shows the user's path and some system information at the top.

```
[shmlog@hydra ~]$ . dot.platform.sh.linux
Fr 28. Okt 18:41:02 CEST 2016
set basedir
set platform
set path
Fr 28. Okt 18:41:02 CEST 2016
[shmlog@hydra ~]$ cd java
[shmlog@hydra java]$ ls
bin fedora ibm oracle readme.txt src
[shmlog@hydra java]$ cd bin
[shmlog@hydra bin]$ ls
compile_jni_stub.sh      create_header.sh  getfrommain.sh
compile_to_class_package_version.sh  create_jni_lib.sh  start_hello.sh
compile_to_class.sh        dot.java.sh       start_package_hello.sh
copyjnilib.sh              dot.java.sh.linux  start_package_log.sh
create_header_package_version.sh  dot.java.sh.mingw  wrong_create_header_package_version.sh
[shmlog@hydra bin]$
```

Illustration 35: The java bin with its scripts

We have here one script that is interesting now. The rest becomes clear when it comes to a real build cycle. So for now a short info about the scripts

- `compile_jni_stub.sh`  
The main build script – its the same for the jni layer as the `makeall.sh` for the C module.
- `compile_to_class_package_version.sh`  
The helper to compile the java class files to byte code. This is a helper for now, but in case you need it you can start it regular itself.
- `compile_to_class.sh`  
The first version to do it – now only for documentation purpose.
- `copyjnilib.sh`  
The helper to copy the jni bridge lib in place for testing.
- `create_header_package_version.sh`

The helper to create the C header from the byte code file of the jni bridge class.

- `create_header.sh`

The first header creation script, now only for documentation purpose.

- `create_jni_lib.sh`

The helper to compile the bridge C code and link to the library to get – ahem – the library ? I think here we have to distinguish the library ( C module ) and the jni bridge library for the jni stuff. We call the later from now on the jni library.

- `dot.java.sh`

The environment settings for the other scripts. Its the same for the java build as the `dot.platform.sh` for the C module.

- `dot.java.sh.linux`

OK, already done here.

- `dot.java.sh.mingw`

OK, already done here.

- `getfrommain.sh`

The script to transfer the headers and the library to all vendor specific jdk directory's. This is the real thing here, see below.

- `start_hello.sh`

Helper to start the jni thing for the first tests, now only here for documentation purpose.

- `start_package_hello.sh`

Another early bird. Simply for documentation purpose.

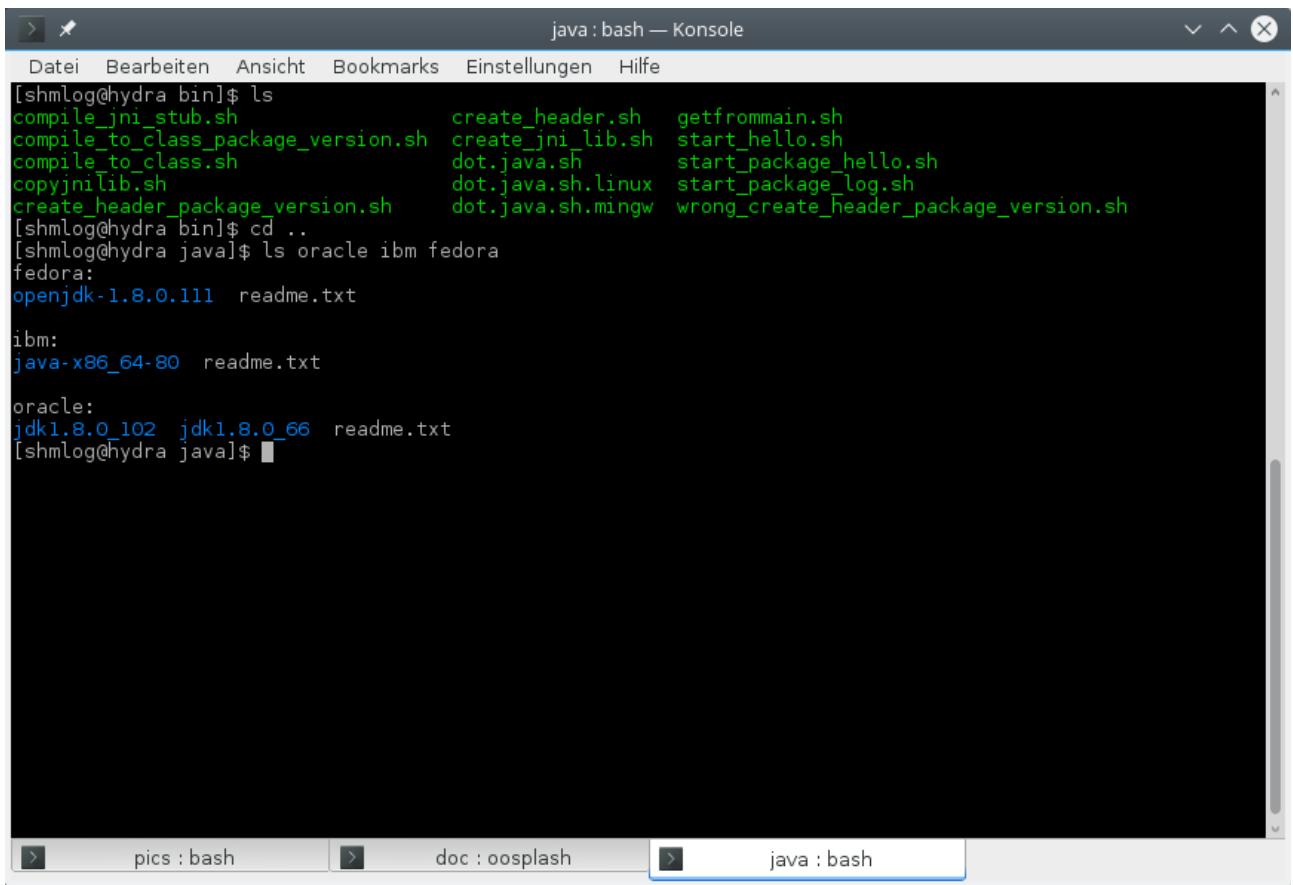
- `start_package_log.sh`

The helper to start the Test class main methods. This is the real thing – see below.

- `wrong_create_header_package_version.sh`

A gag – how to do it wrong ... or : when a C programmer tries to do things logically with java ...

OK. There were two real things, but only one for now. The `getfrommain.sh`. It transfers the headers and the library to the mystery vendor and jdk directory's. We see now for names like oracle and IBM and fedora, its time for a peek here...



A screenshot of a terminal window titled "java : bash — Konsole". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". The terminal shows the following command-line session:

```
[shmlog@hydra bin]$ ls
compile_jni_stub.sh      create_header.sh  getfrommain.sh
compile_to_class_package_version.sh  create_jni_lib.sh  start_hello.sh
compile_to_class.sh        dot.java.sh       start_package_hello.sh
copyjnilib.sh              dot.java.sh.linux  start_package_log.sh
create_header_package_version.sh  dot.java.sh.mingw  wrong_create_header_package_version.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra java]$ ls oracle ibm fedora
fedora:
openjdk-1.8.0.111  readme.txt

ibm:
java-x86_64-80  readme.txt

oracle:
jdk1.8.0_102  jdk1.8.0_66  readme.txt
[shmlog@hydra java]$
```

*Illustration 36: The vendor's directories*

So that's now the vendor things, and inside the jdk things.

You should now check for the best match and make a clone with the proper naming for vendor and jdk.

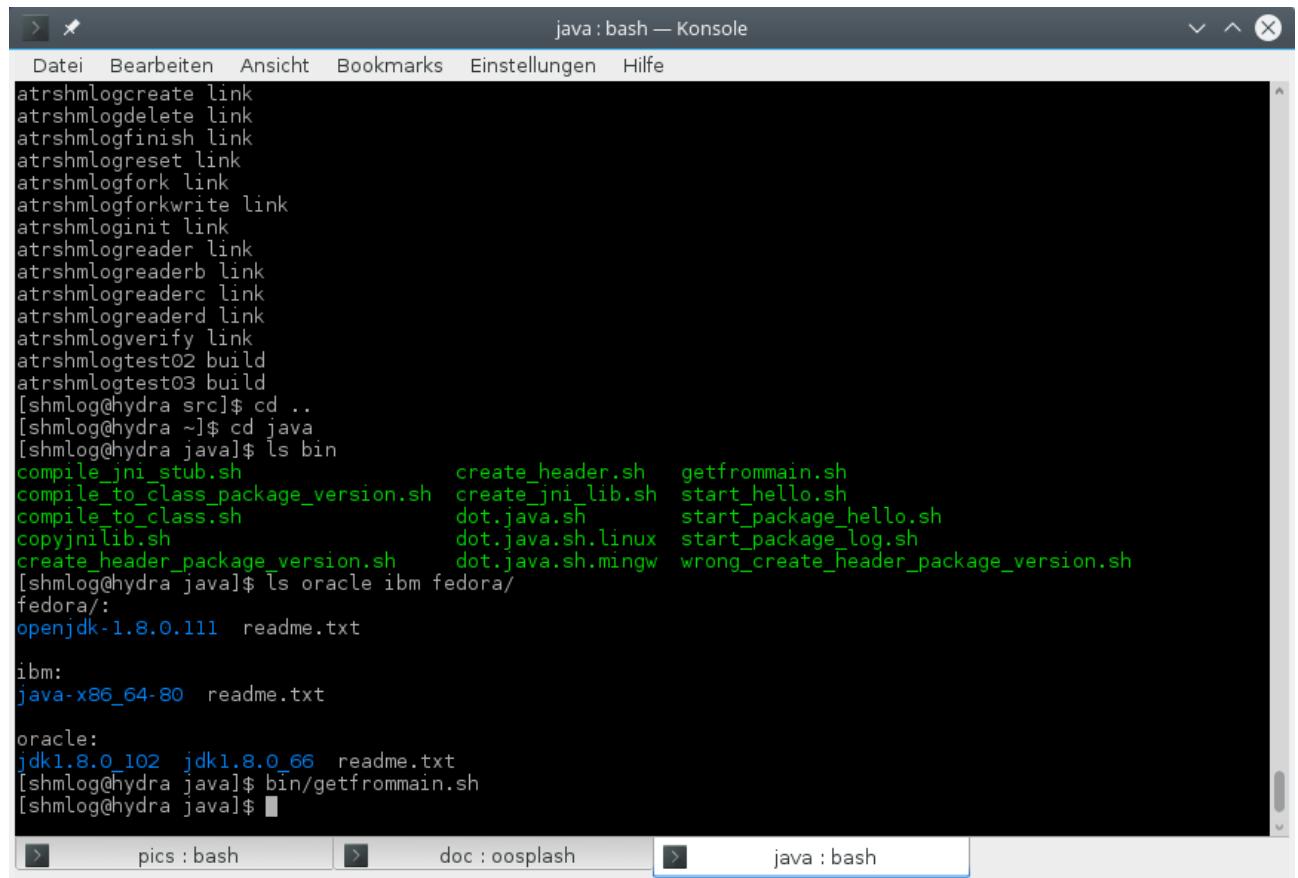
OK. So we transfer with getfrommain.sh from the src what's there to the whole bunch.

Meaning: you should have the real headers in src in place. And the real library. Don't mix platforms or versions – then you have to ignore that script.

If everything is right we can execute the script now. Its mandatory that you are in the java directory for its execution. So we call it relative with bin/getfrommain.sh.

## **Copy headers and lib from the C module**

And so we do it



The screenshot shows a terminal window titled "java : bash — Konsole". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". Below the menu is a list of files and directories transferred from the source directory:

```
atrshmlogcreate link
atrshmlogdelete link
atrshmlogfinish link
atrshmlogreset link
atrshmlogfork link
atrshmlogforkwrite link
atrshmloginit link
atrshmlogreader link
atrshmlogreaderb link
atrshmlogreaderc link
atrshmlogreaderd link
atrshmlogverify link
atrshmlogtest02 build
atrshmlogtest03 build
[shmlog@hydra src]$ cd ..
[shmlog@hydra ~]$ cd java
[shmlog@hydra java]$ ls bin
compile_jni_stub.sh      create_header.sh  getfrommain.sh
compile_to_class_package_version.sh  create_jni_lib.sh  start_hello.sh
compile_to_class.sh        dot.java.sh       start_package_hello.sh
copyjnilib.sh              dot.java.sh.linux  start_package_log.sh
create_header_package_version.sh    dot.java.sh.mingw  wrong_create_header_package_version.sh
[shmlog@hydra java]$ ls oracle ibm fedora/
fedora/:
openjdk-1.8.0.111  readme.txt

ibm:
java-x86_64-80  readme.txt

oracle:
jdk1.8.0_102 jdk1.8.0_66  readme.txt
[shmlog@hydra java]$ bin/getfrommain.sh
[shmlog@hydra java]$
```

At the bottom of the terminal window, there are three tabs: "pics : bash", "doc : oosplash", and "java : bash". The "java : bash" tab is currently active.

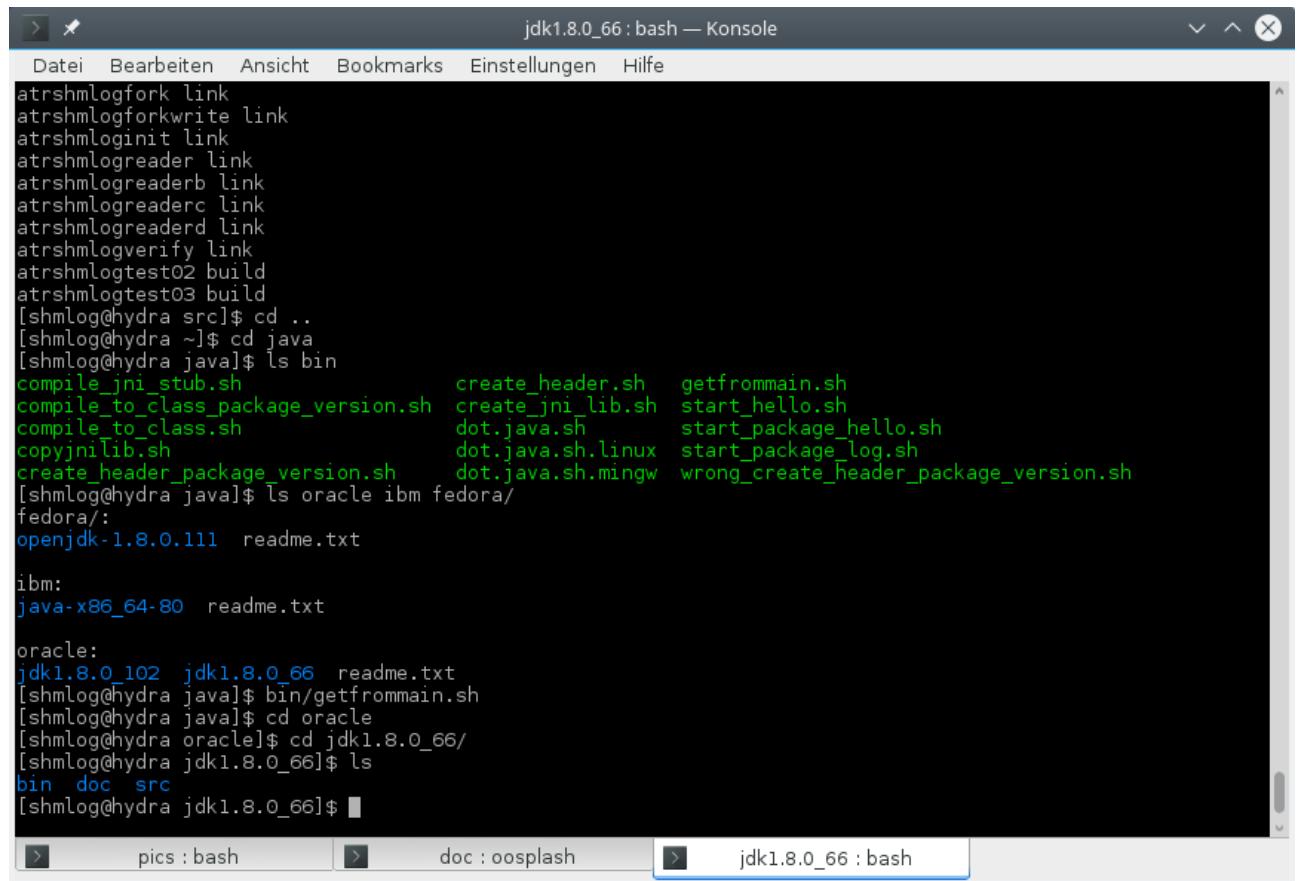
*Illustration 37: Transfer from BASEDIR/src to the vendor's directories*

Not much noise. Well, we will see....

Now its time to check for a vendor and the jdk directory's. We use here the first implementation, that's the oracle and the jdk1.8.8\_66.

## **Change into your vendor and jdk directory**

We change and then see for it



The screenshot shows a terminal window titled "jdk1.8.0\_66 : bash — Konsole". The window contains a list of files and directories. The files include "atrshmlogfork link", "atrshmlogforkwrite link", "atrshmloginit link", "atrshmlogreader link", "atrshmlogreaderb link", "atrshmlogreaderc link", "atrshmlogreaderd link", "atrshmlogverify link", "atrshmlogtest02 build", "atrshmlogtest03 build", "[shmlog@hydra src]\$ cd ..", "[shmlog@hydra ~]\$ cd java", "[shmlog@hydra java]\$ ls bin", "compile\_jni\_stub.sh", "compile\_to\_class\_package\_version.sh", "compile\_to\_class.sh", "copyjniLib.sh", "create\_header\_package\_version.sh", "[shmlog@hydra java]\$ ls oracle ibm fedora/", "fedora/:", "openjdk-1.8.0.111 readme.txt", "ibm:", "java-x86\_64-80 readme.txt", "oracle:", "jdk1.8.0\_102 jdk1.8.0\_66 readme.txt", "[shmlog@hydra java]\$ bin/getfrommain.sh", "[shmlog@hydra java]\$ cd oracle", "[shmlog@hydra oracle]\$ cd jdk1.8.0\_66/", "[shmlog@hydra jdk1.8.0\_66]\$ ls", "bin doc src", "[shmlog@hydra jdk1.8.0\_66]\$". Below the terminal window, there are three tabs: "pics : bash", "doc : oosplash", and "jdk1.8.0\_66 : bash", with "jdk1.8.0\_66 : bash" being the active tab.

*Illustration 38: Inside a vendor dierctory...*

Not a real surprise. A bin – with scripts – a doc – with documentation – and a src.

Let's see for it.

The content of the three

A screenshot of a terminal window titled "jdk1.8.0\_66 : bash — Konsole". The window shows the following directory structure and files:

```
ibm:  
java-x86_64-80  readme.txt  
  
oracle:  
jdk1.8.0_102  jdk1.8.0_66  readme.txt  
[shmlog@hydra java]$ bin/getfrommain.sh  
[shmlog@hydra java]$ cd oracle  
[shmlog@hydra oracle]$ cd jdk1.8.0_66/  
[shmlog@hydra jdk1.8.0_66]$ ls  
bin  doc  src  
[shmlog@hydra jdk1.8.0_66]$ ls bin doc src  
bin:  
compile_jni_stub.sh      create_jni_lib.sh  
compile_to_class_package_version.sh  dot.java.sh  
compile_to_class.sh       start_hello.sh  
copyjnilib.sh            start_package_hello.sh  
create_header_package_version.sh    start_package_log.sh  
create_header.sh          wrong_create_header_package_version.sh  
  
doc:  
Java Native Interface (JNI) - Java Programming Tutorial-Dateien  
Java Native Interface (JNI) - Java Programming Tutorial.htm  
  
src:  
atrshmlogjnipackage.c      create_jni_lib.sh      includes  
compile_jni_stub.sh        de                      libatrshmlog.a  
compile_to_class_package_version.sh  dot.java.sh  
compile_to_class.sh         hellojni.h           myjnipackage  
copyjnilib.sh              HelloJNI.java        start_hello.sh  
create_header_package_version.sh  hellojnitest01.c  start_package_hello.sh  
create_header.sh            hellojnitestpackage01.c  start_package_log.sh  
[shmlog@hydra jdk1.8.0_66]$
```

The terminal window has tabs at the bottom: "pics : bash", "doc : oosplash", and "jdk1.8.0\_66 : bash".

Illustration 39: ... and more inside of it.

OK. The bin seems to be a copy of the scripts – but in fact its an adapted copy – see later on.

The doc contains a nice little documentation of the basics in jni – if you are interested and have to do it by yourself you can check it. Its somewhat outdated for the examples, but it helps.

The src is the real thing. We have again the – adapted - scripts here, so we are free to change them if we need. For reference there are the scripts in bin, so don't change them in bin till you have done the whole thing. Start only in src with changing.

We switch to src and now comes the list of the files in there.

- compile\_jni\_stub.sh
  - As already said. The main build script.
- compile\_to\_class\_package\_version.sh
  - The helper for byte code generation.
- compile\_to\_class.sh
  - Ignore it.
- copyjnilib.sh
  - The helper to copy the jni library.

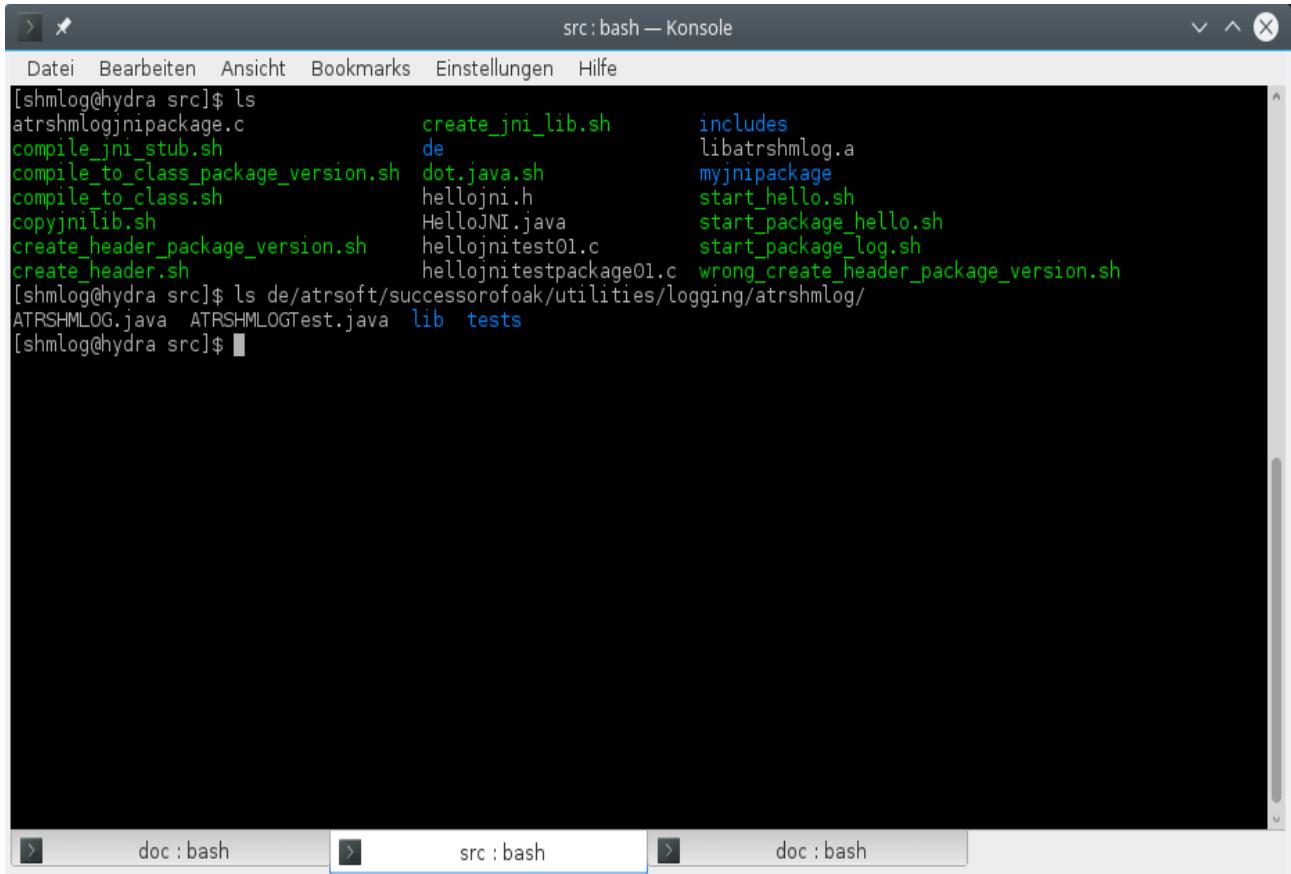
- `create_header_package_version.sh`  
The helper to create the C header.
- `create_header.sh`  
Ignore it.
- `create_jni_lib.sh`  
The helper to compile the bridge and link.
- `dot.java.sh`  
The setting of the environment variables. Next thing to do.
- `hellojni.h`  
The header for the simple example code.
- `HelloJNI.java`  
The class for the simple test code.
- `hellojnitest01.c`  
Test code for the simple test code.
- `hellojnitestpackage01.c`  
Test code with a package.
- `libatrshmlog.a`  
The copy of the library from the C module (check time stamp and size and whatever you need too ... should be from BASEDIR/src )
- `start_hello.sh`  
Helper to start the simple test.
- `start_package_hello.sh`  
Helper to start the simple package test.
- `start_package_log.sh`  
Helper to start the Test class main in the package tree.

For the directory's : includes is clear – check for the files and for the time stamp and length of the headers from BASEDIR/src .

For de its the start of the package path. If you are a java developer you know what follows.

Check its end ...

## The package directory



A screenshot of a terminal window titled "src : bash — Konsole". The window shows the output of the "ls" command in the "src" directory. The files listed include: atrshmlogjnipackage.c, compile\_jni\_stub.sh, compile\_to\_class\_package\_version.sh, compile\_to\_class.sh, copyjniLib.sh, create\_header\_package\_version.sh, create\_header.sh, de, dot.java.sh, hellojni.h, HelloJNI.java, hellojnitest01.c, hellojnitestpackage01.c, includes, libatrshmlog.a, myjnipackage, start\_hello.sh, start\_package\_hello.sh, start\_package\_log.sh, and wrong\_create\_header\_package\_version.sh. Below this, it shows the path: atrshmlog@hydra src]\$ ls de/atrsoft/successorofoak/utilities/logging/atrshmlog/. The bottom of the window shows tabs for "doc : bash" and "src : bash".

Illustration 40: Inside the package

Not very much.

- ATRSHMLOG.java  
The jni bridge class.
- ATRSHMLOGTest.java  
The test class – for now its the only test I have, so don't be surprised if more follows in the future.
- Lib  
The directory that contains the jni library for test.
- Tests  
the module test package.

OK. Now that we know that tree, check for the others.

When you are ready, you can read on.

Ready ? So fast – well, if you said so.

Every vendor has the jdk directory, and every jdk has the same structure. So our little helper getfrommain.sh is not so complicated after all.

It simply copy in every src directory the lib and in every includes the two include files.

Helps to do it manually. If you need.

The real different thing in every jdk is for now the dot.java.sh. It contains the settings of JAVA\_HOME, the JAVA\_OS and so changes from jdk to jdk. And from platform to platform.

If you check the jdk and its include directory, you will find the platform named directory for the stuff that is not identical for platforms.

So try to think backwards : if you have a new vendor and jdk you clone a nearby existing one, check for the JAVA\_HOME, adapt that in the dot.java.sh, check for the directory in the jdk's include directory, adapt that to the JAVA\_OS and with a little luck you are done.

The only thing left is the name of the library, and that's in the create\_jni\_lib.sh for the existing platforms.

So much for a complex build problem....

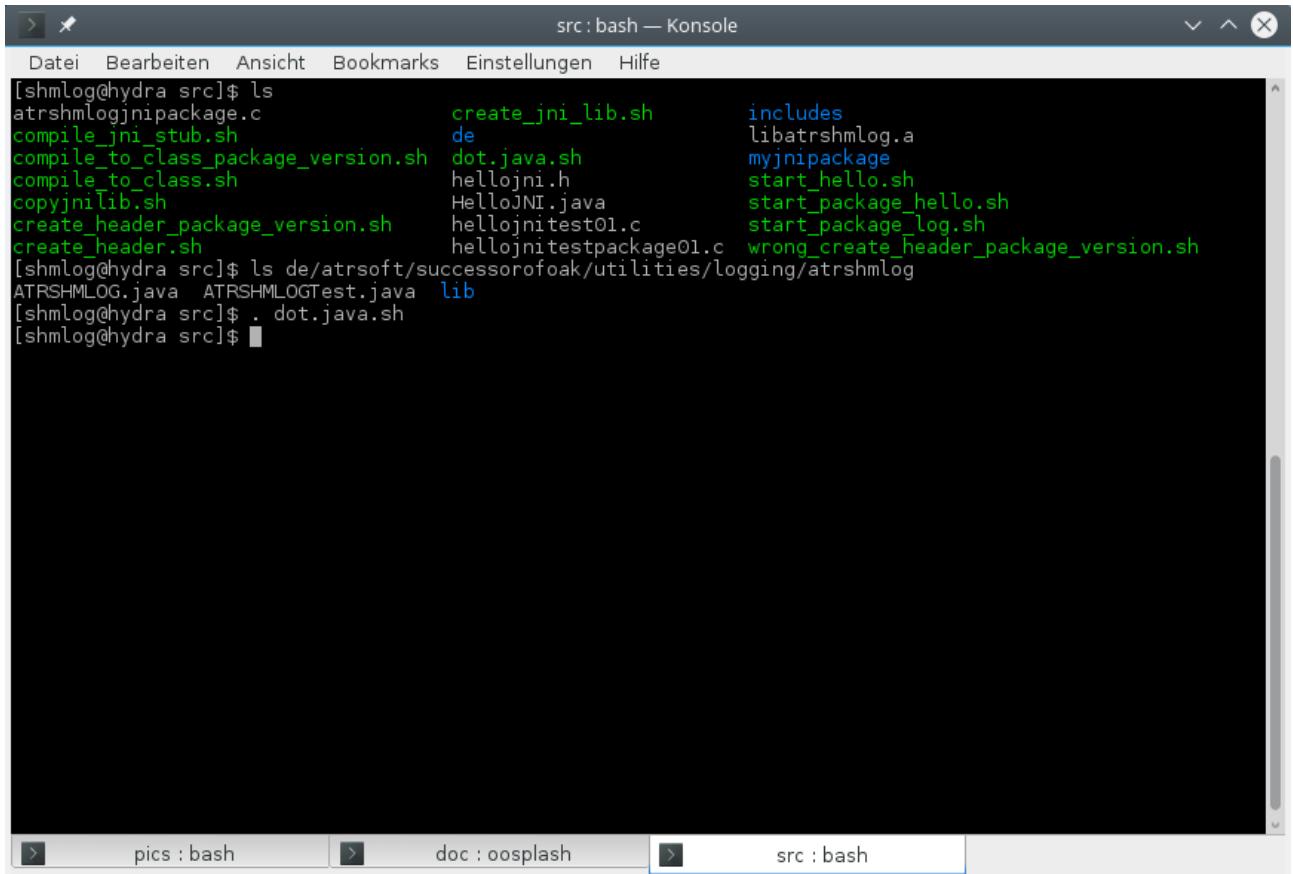
Its now time to do the thing.

## **Setting the environment**

First you have to start with the platform settings from the C module.

Source the proper dot file in BASEDIR. Then switch to the java src again.

We then simply source dot.java.sh



The screenshot shows a terminal window titled "src : bash — Konsole". The menu bar includes "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". The terminal output shows the user navigating through directory structures and running shell scripts to set up the build environment. The user runs "ls" to list files, "cd" to change directories, and ". dot.java.sh" to source the configuration script. The terminal window is part of a larger interface with tabs for "pics : bash", "doc : oosplash", and "src : bash".

```
[shmlog@hydra src]$ ls
atrshmlogjnipackage.c      create_jni_lib.sh      includes
compile_jni_stub.sh         de                      libatrshmlog.a
compile_to_class_package_version.sh dot.java.sh      myjnipackage
compile_to_class.sh          hellojni.h            start_hello.sh
copyjniLib.sh                HelloJNI.java        start_package_hello.sh
create_header_package_version.sh hellojniTest01.c   start_package_log.sh
create_header.sh              hellojniTestpackage01.c wrong_create_header_package_version.sh
[shmlog@hydra src]$ ls de/atrsoft/successorofoak/utilities/logging/atrshmlog
ATRSHMLOG.java  ATRSHMLOGTest.java  lib
[shmlog@hydra src]$ . dot.java.sh
[shmlog@hydra src]$
```

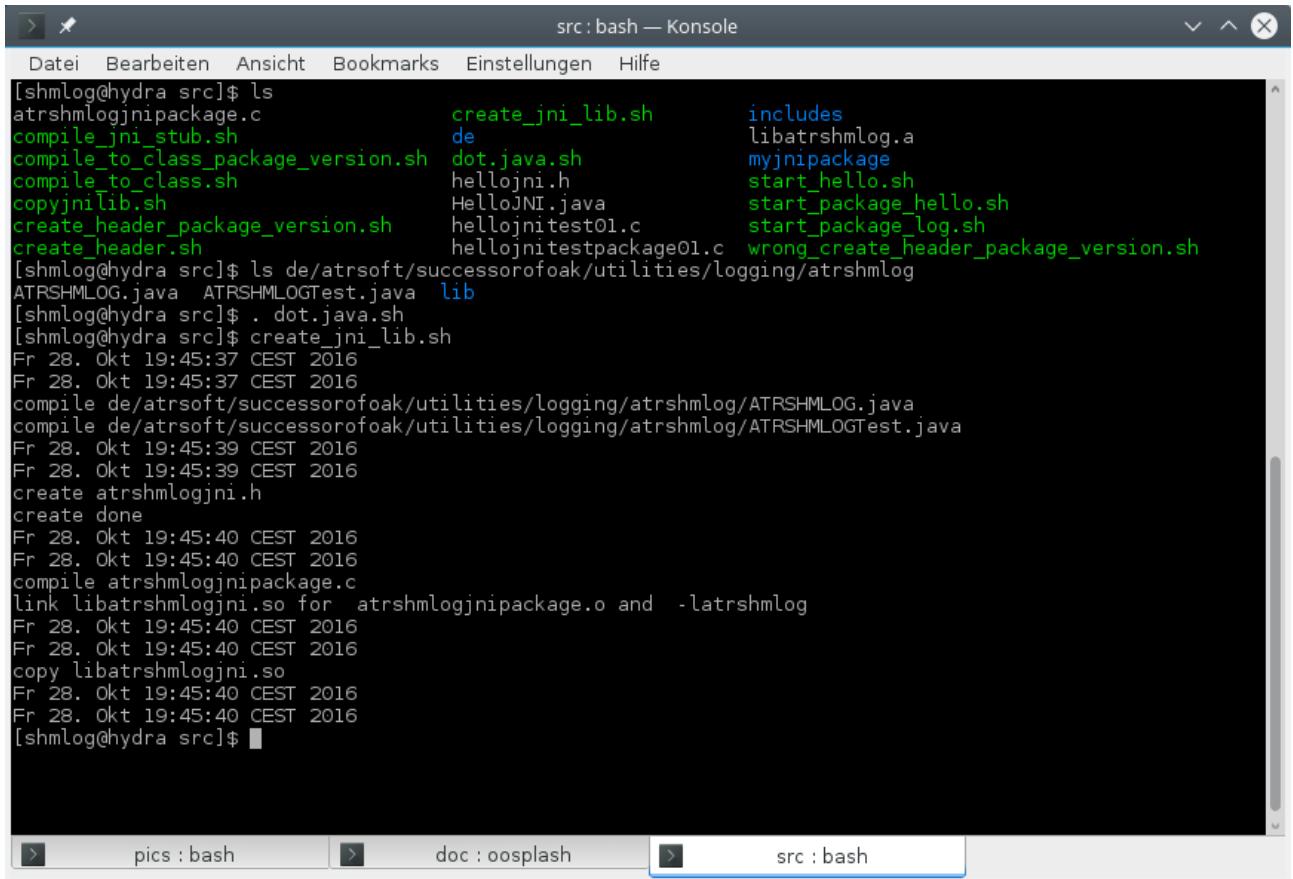
*Illustration 41: Setting up the build environment*

No noise here. If you insist you can check the environment.

Next is to start the build.

## **Building with *create\_jni\_lib.sh***

We start the script



The screenshot shows a terminal window titled "src : bash — Konsole". The terminal displays the execution of a shell script named "create\_jni\_lib.sh". The output shows various commands being run, including "ls", "dot.java.sh", "create\_jni\_lib.sh", and "javac". The terminal window has a menu bar at the top with German labels: Datei, Bearbeiten, Ansicht, Bookmarks, Einstellungen, Hilfe. Below the menu is a toolbar with icons for copy, paste, and others. The bottom of the window shows tabs for "pics : bash", "doc : oosplash", and "src : bash", with "src : bash" currently selected.

```
[shmlog@hydra src]$ ls
atrshmlogjnipackage.c           create_jni_lib.sh      includes
compile_jni_stub.sh              de                      libatrshmlog.a
compile_to_class_package_version.sh dot.java.sh        myjnipackage
compile_to_class.sh               hellojni.h       start_hello.sh
copyjniLib.sh                   HelloJNI.java    start_package_hello.sh
create_header_package_version.sh hellojnitest01.c   start_package_log.sh
create_header.sh                 hellojnitestpackage01.c wrong_create_header_package_version.sh
[shmlog@hydra src]$ ls de/atrsoft/successorofoak/utilities/logging/atrshmlog
ATRSHMLOG.java  ATRSHMLOGTest.java  lib
[shmlog@hydra src]$ . dot.java.sh
[shmlog@hydra src]$ create_jni_lib.sh
Fr 28. Okt 19:45:37 CEST 2016
Fr 28. Okt 19:45:37 CEST 2016
compile de/atrsoft/successorofoak/utilities/logging/atrshmlog/ATRSHMLOG.java
compile de/atrsoft/successorofoak/utilities/logging/atrshmlog/ATRSHMLOGTest.java
Fr 28. Okt 19:45:39 CEST 2016
Fr 28. Okt 19:45:39 CEST 2016
create atrshmlogjni.h
create done
Fr 28. Okt 19:45:40 CEST 2016
Fr 28. Okt 19:45:40 CEST 2016
compile atrshmlogjnipackage.c
link libatrshmlogjni.so for atrshmlogjnipackage.o and -latrshmlog
Fr 28. Okt 19:45:40 CEST 2016
Fr 28. Okt 19:45:40 CEST 2016
copy libatrshmlogjni.so
Fr 28. Okt 19:45:40 CEST 2016
Fr 28. Okt 19:45:40 CEST 2016
[shmlog@hydra src]$
```

*Illustration 42: Create the jni library*

And there we are.

Two compiles java classes, one new header creation. One compile and one link line. Finally the copy to the test directory.

All what's left is the test.

## Testing the jni bridge

We have first to create a shared memory buffer with atrshmlogcreate. I will use 4711 and 8 for it. Then the init for the area. Next the test of the bridge. Then the reader and the convert.

At last we can check for the result log.

And because it was already done for the C test program we do it in short form now.

The screenshot shows a terminal window titled "src : bash — Konsole". The terminal displays the following command-line session:

```
[shmlog@hydra src]$ cd
[shmlog@hydra ~]$ cd src
[shmlog@hydra src]$ atrshmlogcreate 4711 8
shm log create.
logsystem version is 1.
ipckeys is 0X1267
count of buffer is 8

paste this into the user process environment startshell
or into the profile or ENV file

ATRSHMLOG_ID="5767181"
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
[shmlog@hydra src]$ . dot.atrshmlog
[shmlog@hydra src]$ atrshmloginit
shm log attach and init.
logsystem version is 1.
[shmlog@hydra src]$ cd ../../java/oracle/jdk1.8.0_66/src/
[shmlog@hydra src]$ start_package log.sh
logging done. Times start 31941386702690 times end 31941386613570 return attach is 0 return write is 0
[shmlog@hydra src]$ cd
[shmlog@hydra ~]$ cd src
[shmlog@hydra src]$ atrshmlogstopreader
shm log attach and set reader flag and pid.
logsystem version is 1.
pid before 0
flag before 0
[shmlog@hydra src]$ atrshmlogreaderd d1
bash: atrshmlogreaderd: Befehl nicht gefunden...
Die Suche nach der Datei ist fehlgeschlagen: curl error (6): Couldn't resolve host name for https://www.virtualbox.org/download/oracle_vbox.asc [Could not resolve host: www.virtualbox.org]
[shmlog@hydra src]$ atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
logging done.
writer data : time 994600 count 1 per use 994600
[shmlog@hydra src]$ atrshmlogconv d1
shm log converter from file 'd1/0/atrshmlog_p14549_t14550_f0.bin' to file 'd1/0/atrshmlog_p14549_t14550_f0.txt'.
id 1 acquiretime 1130 pid 14549 tid 14550 slavetime 1600 readertime 10500 payloadsize
1031 shmbuffer 0 filenumber 0
[shmlog@hydra src]$ cat d1/0/atrshmlog_p14549_t14550_f0.txt
0000014549 000000000014550 000 00000000000000000000 000031941386613040 000031941386613570 000000000000000530
1477677102446473969 1477677102446474168 0000000000000000199 00000000001 I 00000000001
0000014549 0000000000014550 000 00000000000000000000 000031941386702690 000031941386703410 000000000000000720
1477677102446507620 1477677102446507890 0000000000000000270 00000000002 I 00000000001
0000014549 0000000000014550 000 00000000000000000000 000031941388264210 000031941388641260 000000000000377050
1477677102447093759 1477677102447235290 000000000000141531 00000000003 i 0000000001 loop 1000 times gettime
0000014549 0000000000014550 000 00000000000000000000 000031941388743710 000031941388743890 000000000000000180
1477677102447273746 1477677102447273813 0000000000000000067 0000000004 I 00000000001
```

Illustration 43: Test of the jni bridge

HM. One glitch, but a typo is OK at least, the rest worked as expected.

## **Details**

Now we can go in some detailed check for the jni class. For the use of it see the Test class. For the use of additional features you need to check the documentation of the methods. But that's the usual way to do it today after all. You can use the HTML for it, the class is in there or your favorite IDE and javadoc.

If you are interested in the way it works – see the C source or read the chapter about the glory details.

But for now some info from a non java programmer to the java code.

Open the ATRSHMLOG.java with your favorite text editor.

After some brubbling comments you touch the first thing of interest in line 70.

We load the bridge library here.

If you ever need to do that different you can change it. But try not to touch the native code – then you can use it still in place with the existing library. No need for a recompile.

Next stop is after the enum stuff. I did it because it feels right to have some enum's for this kind of stuff. Error codes is natural, and for the statistics the enum's can be a help if you need them. For the strategy its at least a start to understand them. So much for the enum's. And use them only after you are sure you understand the cost. For example don't make error enum if you encounter the 0 return code for most methods.

That takes us to line 1115. Seems to be some simple int numbers, but its worth the stop. We have the option to log not only C stuff strings ( which would be very poor in a java environment) but also to use a byte array. So you can tell the module in the logging what you want to do with it – have it to be a log of a byte array in C style, or in UCS2 style. And if it is a log for a Point in time – meaning we are having only one valid time stamp for the start or even none – or an interval in time – you guess it, its a start and an end time stamp (which can be a 0 for end, see below).

Ahem. Yes, and there is of course the logging of – Strings.

Here you really need to understand what it means. For the time thing its the same. You log a point – only one time has a meaning and its the point in time – or you log an interval. Same thing.

The real problem is the kind of String you want to get in the result log.

Having a language with a 2 byte per character string model makes it natural to log that strings. And so you can do that with the ucs2 constants. One for point, one for interval.

If you use C style you get an internal conversion in the jni bridge code from ucs2 to the thing that jni delivers as UTF string representation. That means your C buddy's have no problems to read the stuff – if they have an UTF conform environment.

That's nice to the C guys, but has a big burden – its slow and happens in the logging phase IN

## YOUR PROGRAM .

So I recommend to use that only as a last resort if things begin to become complicated with guys like project managers or architects or quality assurance or.. or .. or...

If it is really a problem : you can log strings into UTF. And its the jni that convert them. So that should be trustworthy for any guy of the former list.

If you want to have speed, well, then the ucs2 it is.

You have a potential problem with the actual version of the converter, because it then tries to make some silly simple conversion of it. But you get it one to one in the binary file.

You can try to develop a better converter, or write one in java – that's up to you.

For now – you need these four constants because they are hard coded into the module and into the converter. So use them. End of message ...

Now we enter the methods code. I will focus on those of interest, for the others simply check the doc in the class or in the module.

First stop is attach, line 1156.

You use it to connect to a valid area in shared memory. The thing can be created by a tool like atrshmlogcreate and initialized by atrshmloginit, but you can also use the layers create and initShmLog methods – didn't try the later thing for now, so sent me a post card when it works.

Without the connect – no logging.

I suggest you make at least one run without a connect – should be running at full speed and without problems.

OK.

Where do we get the parameters for the attach from ?

Well, that's already said in the C module section – but I know you don't really read that – so I suggest you read the chapter about the environment variables and the attach at least.

Next is the gettimeofday. Its the heart of the timing things, so let's see for the thing.

We have a 64 bit click counter timing model – I know, you didn't read the C module part ....

So we use them here as values. And use a low level long for it. No class, no fancy Long – a simple long. And that will be my last word here. No high level things. I don't need them. If you need them – use another log, it will be slow enough for your needs, I hope.

There is a static method for gettimeofday too. Its sg\_gettime. If you test and you can find its faster in let's say, more than 25 % then the object method please sent me a post card and I will do the static thing at least for the write too.

Speak of the devil. Next is write in line 1248.

This is the reduced form, only core info, no payload at all. This can help if speed is all you need.

Next is the write with String. Please keep in mind that the simple thing has an deep inside buried mechanism to make sense of point in time and interval – which I have only added to reduce jni calls – and that it is capable of doing ucs2 and UTF. So you have to know what you want and use the correct event flag. See for the constants above.

Line 1407 comes first to a surprise – why a second write with a String ?

To make a long story short: This is doing ucs2 only, and it is doing only part of the string – you can give the number of chars that are used from the beginning. So you don't have to make complicated String operations to build one that fits – if you have something big like a stack trace or a json package you can put it in and simply set a limit – no need to build up a new one. If the limit is too big the real length is used.

That's of course only half the thing. You can again log point in time or interval with it.

Line 1394 we have the third time write for a string, and this time you can give the startindex and the length. This is for logging the interesting part of big strings.... You can use it like the second write, only the startindex is needed here also in place and you can log the famous last part of your stack trace or json or whatever you need.

Line 1582 is then the write for a byte array – so you can log everything you are capable to put in such a thing. Its converted in the standard converter as a C string or an ucs2 string. So to make use of it its best to make it for a separate converter of yours. The starting point is the chapter about make a new converter.

Now that we have the core comes the stuff that you perhaps never need – or perhaps do need.

So I try t focus on the details worth here.

Statistics can be helpful. So I build in that thing.

You can get them into an int array. To do that you have to give the array into the get thing, and to do that you need to know how big it is in creation of it – its max index plus one ...

So much for statistics. See the enum's and the atrshmlogstat script for the info you get. Most can be helpful in the adjustment process.

We have a sleep internal in the module, and on most platforms its a nano sleep. So I decided to make it available to you. But be prepared: on some platforms it might not work as expected. See the internet for a discussion on that thing. I give you one hint: fenster;plural is again the bad guy in the game here. End of transmission...

Then comes the environment variable stuff. Can be helpful if you write your own tools, but I doubt you will need it. Perhaps the set prefix, if you like another name for the variables prefix part.

The version starts at line 1685.

To make it simple: There is a major version and it is the compatibility level of the tools with respect to the area layout. Also the reader and in last consequence the converter is in here.

That means that I give a new version if the layout changes.

The minor version is then about new features. That can also mean a new platform – but the layout is not changed here.

Last is the patch version, which is about the bad thing stuff and how we get rid of it. Bugs.

For the major version I plan to do an odd even thing. Odd is a stable, even is an experimental but otherwise checked one.

With line 1720 we enter the event stuff. As you know from the former chapters ... ups, I forgot, you don't read those. Right. Events are the way to say where your logging happens, and so you use them as an info like a line number – but not the same thing, please. So you can say something like “oh yeah, that was at event 2544 and now I understand why it did then event 7536....”. That stuff.

We have implemented an array of chars in C for them. So you have plenty of them (concrete a number of 10000) in the beginning. If you need more you can make it bigger. Costs memory. And because they are related to an location we can use them to switch logging on or off for that specific event. So the char array holds at least an on and off flag.

After the event stuff we have the logging state of the thing. There are flags to suppress it on level of the system ( the so called “Ich habe fertig” flag in the area) and on the process. For a thread you can shut down logging too. And for the final systems shutdown we have another one, so that you can come up against the so called zombie revival problem.

Next is the information about things in and around the shared memory area. That means the version, the buffer count and the flags. Also you can get the address of the area as a long and use it with a bunch of methods.

With 2028 we enter the configuration of the module. We can get and set a bunch of things. There is the buffer size first. Its fixed for the first buffers, but when dynamic allocation comes in we can change it (to be precise : reduce. No growth possible).

We can set the number of slave threads before the attach, and we can change the id of the clock we use in the getClocktime.

Then there is stuff to handle the slaves. You can shut them down if you think its a wise thing to do – they will exit their main loops the next time they hit it. You can set the wait flag for the cleanup for exiting the slaves, and you can set the time a slave waits when it has found out there is no work to do.

Line 1896 we enter the buffer handling. There are buffers for the log for the threads – so much here – and we have a bunch allocated static in advance. Everything else is up to the program - you can set the number of buffers we allocate in one low level alloc, set the initialization of them ( with memset 0 ).

In between – perhaps I should change this - is some other stuff. The initime things. We make at attach a kind of time stamp triple, two clicktime and a real time. Every time we move a buffer to the area it gets the actual triple too. So we can approximate the clock times in the logging info with real time. Of course that depends on a bunch of heuristics, so don't trust it too far. For example on

fenster;plural there is no thing like a get real time with nanos thing. Best approximation is a Filetime with nanos in 100 step width ... did I ever mention I don't like that OS ?

You can get the id of the highest used buffer – which gives you an approximation how much memory is used for logging.

Then there is again stuff in line 2085. We can stop a threads logging. That's done by stop and it is the thread that has to call it for.

We can also flush the buffers. That is a good thing to slow it down – meaning we have after the flush no usable buffer till they have made the round trip to the area and then back. Consider be warned. And perhaps I will even make it worse – build something like an auto flush to help that guys from the debugging frontier division ....

Back to the buffers. An important thing is at 2108 . You can set the threads strategy. That means you can have the thread decide what to do if the buffers are full. In that case we need at last the options to discard or to spin lock. Waiting is also possible, and for highly unpredictable loads I have the adaptive strategy's in place. See the glory details for that. And the adjustment chapter.

In 2147 we do the final thing for slaves – we create one if we need. OK, if we create we have also to maintain the opposite case – we kill one and then we have to balance the count with decrement.

The fences stuff follows. There are 13 thread fences build into the thing. See the glory details for the story behind them. You can switch them on if you need. That's for the case you encounter inconsistency for the log. Can happen. Had it in development two times. And for different platforms it can be worse. So they are off for the Intel boxes, but you can switch them on.

One method to get a real time, if it were not for the need to check this against your internal clock....

Now comes some odd stuff. We need to talk about a forced thread death here.

First in line 2393 a method that the thread has to execute. It delivers the address of the thread local storage ( to be precise : we use a struct and that is the start address).

With the next you get the thread id – the thing is the best I could get, so be happy if it matches your thread system thread identification thing. If not you have to use the tid or the thread local address and match it to the thing you get from your system.

Now to work: line 2410 makes a stop for a thread – and this time its with the threads thread local address done – so you can use it to turn down operation for a thread from your application. This works for a logging thread and a slave thread.

For the logging thread the buffers are dispatched to the area, and you never again make a log for it. No way back. The buffers are recycled after that and no longer the thread can use them.

For the slave the next activation will hit the end switch and the slave exits after the cleanup. So this is no kill, but a graceful and normally fast stop for the slave.

We skip the already mentioned init buffer stuff here.

Line 2522. Next is the iteration method for the slave list. Well, you need a way to identify the slave,

and you simply get no info from its start that's worth it. Perhaps I will change the interface in future, but now there is none. So you use the iteration and start with a null and get the first slave's local – a piece of memory for the slave only.. Call it with it and you get the next. Till end of story, then you get a null.

So you can pick a slave and use this together with the next method to get its tid. The rest is a call to turn off.

That's the simple and clean way. Now for the dirty way.

You need to kill the slave – why I don't know – so you have it here. Retrieve the slave local. Get the tid. Identify your target. Remove it from the slave list (next method) and then kill the slave.

You need to kill a thread (this time not a slave) - why I don't know – so you have it here. Retrieve the thread local. Get the tid. Identify your target. Then kill the thread. Use the reuse and the tid of the module to reclaim the buffers of the thread.

You will lose the buffers for recycle, the thread statistics, and the thread local will no longer be valid. So you cannot give the buffers free, but the cleanup at end will move them to the area at least.

There is now the reuse thread buffers method, so you can after the kill is done recycle the buffers if you have the tid of the thread – from the point of the module. Still you will lose the last statistics of it.

That's dirty, and if you misunderstood something best dangerous, likely fatal for your program. So be sure you know that you need that. I prefer the clean way. But you can do it...

At line 2568 we have the first administrative function of the module – verify. Its normally only part of the higher level so its not for normal clients. If you want to make all for the java things you can do this. Make your own verify.

Next is the clicktime via clock id . That's needed if you have different clock models or different OS calls. For now its simply a layer of the low level click time functions.

The next is read and readFetch. To make a long story short: you can write your own reader. The backbone functions are here. The interface is a bit odd at least. An array to put the stuff is OK, but an array for the secondary info's is at least odd. But its simple, fast, and I think I need no difficult oversize getter stuff then – don't forget that a reader has to be VERY performant.

For the read its the somewhat outdated call, use the readFetch instead.

Next is in line 2818 the create. So you can make your own buffers in a test driver or make your own version of the create tool.

This has the same consequences as for the module on the mingw port – and I guess for all the ways to handle the things on the fenster;plural system. Read the glory details for it or see the mingw port .

Next is the destroy of the buffer. OK, cleanup and init so we can mimicry the delete, init and finish tools.

Last are two little helpers to be used to make a thing like the signal tools or the dump or the defect

tool. They use the raw C pointer thing and you should NOT use them in any other thing like a valid area buffer.

OK, this was a fast trip through the thing.

Let me summarize it.

We can create, delete, init, deinit, inspect and switch flags for the area.

We can attach to it and then do things like read and write.

We can configure the parameters of the module, which are buffer related, and slave thread behavior.

We can inspect the environment – partly, only the variables here – and the initial times, real time and clicktimes.

And we can switch on and off the fences if we need them.

That's a lot more than a simple “open, write, flush, close” thing. So don't be surprised that it takes some time to master it. For the start check the Test class, its perhaps all you will ever need.

## The python language support

After I had done the initial jni stuff it was clear for me to make it to other languages too.

So on my first list I had python, perl and for those others SWIG in case they use that.

For the python I had to start at ground zero.

I am a newbie in python, so I first had to take some basic stuff from the python documentation page about the making a C module work in python thing.

OK, that's not too hard. Making spam for the 3.6 was helpful.

And then I made it in rough 2 hours to a first test drive. The thing attached, took some times and wrote the technical log. WOW.

Next was to build all functions that are now the interface functions in my usage of terms here.

And again it worked out, I could even read and read\_fetch things. NICE.

That was about 6 hours later.

Then I switched back and checked for the two things I had found in the spam that I wanted to support, the embedded thing – well, this was easy, nothing to do for me here – and the C API capsule thing.

This took again rough 10 hours and I had in the final state the core functions too.

So again some swift changes and several builds and I was there.

Next was to make a big review round, and after two hours I had it done so far.

Last thing was the thing you most likely already know – the thing about python sex.

## Python's SEX

Well, all living things on earth seem to have that reproduction mechanism build in.

And this is also true for the mammal, reptile, bird, fish, even plants.

For the higher ordered it is always done by the mix of two individuals of different gender.

And so there is the male and female thing.

For the python's its different.

They have the gender 2 and 3.

Gender 2 is the python 2 way, and gender 3 is the python 3 way.

And they simply don't mix. Its not clear for me how the python's then reproduce, but that's the facts.

They simply don't mix and still the genders grow bigger and bigger .....

So I had to support not only the capsule C API, but also the two different genders 2 and 3.

OK, some macro stuff did the job, and so you can do it in the 2 way or the 3 way., and it was done in about an hour of my time with tests.

## How it works

The module is a direct used implementation. Under cover you use the C module. The module is capable of doing its job simply by using the internal infrastructure as in a C environment. No need for changes inside.

This means you get in theory the same speed as for the C module itself. The only thing that slows down is the python parameter handling in the bridge code itself.

The rest is more or less simple. Use of the function and no object stuff. Simple raw data types, no classes, no exceptions and no use of the python side with the exception of the exception atrshmlog.error. Barely some string creations.

If you are interested the atrshmlogmodule.c is a good start.

The bridge always uses a so called core function.

The core function itself is used in the C API for the use in other modules. See the header atrshmlogmodule.h for this stuff.

The core function is most of the time only used to convert the types from a call to a C module function to the usual stuff in the python module.

The module gives also the error Object and a small number of constants to the user.

So you have this in module init.

A simple import gets it in. If you made a module for the 2 it must be used by a 2, if it was made for 3 it must be the 3 – any mix and you will encounter problems – best in build phase, worst in the try to load it.

## How to use it

The user needs the following parts:

- Shared library or dll atrshmlog

This is the bridge code from atrshmlogmodule.c and the linked library from the C module, libatrshmlog.a. You need also the compiler support lib's and in case of the mingw port the additional dll 's of the mingw system.

- The support programs

You need the matching C support programs for the basic stuff you don't want to reinvent. Reader, converter, even create and delete. So if you need a pure (?) python only solution you can replace them with your own code python counterparts. But for now I deliver nothing so you should start with the C programs.

So you first have to build it – again I am not delivering binary code, only source, so if you are interested in binary you have to contact me and I will see what I can do. But for now you have to start with the python bridge file atrshmlogmodule.c and atrshmlogmodule.h.

## **How to build it in the first place**

We start with the C module.

After this is in place we have to check our next options.

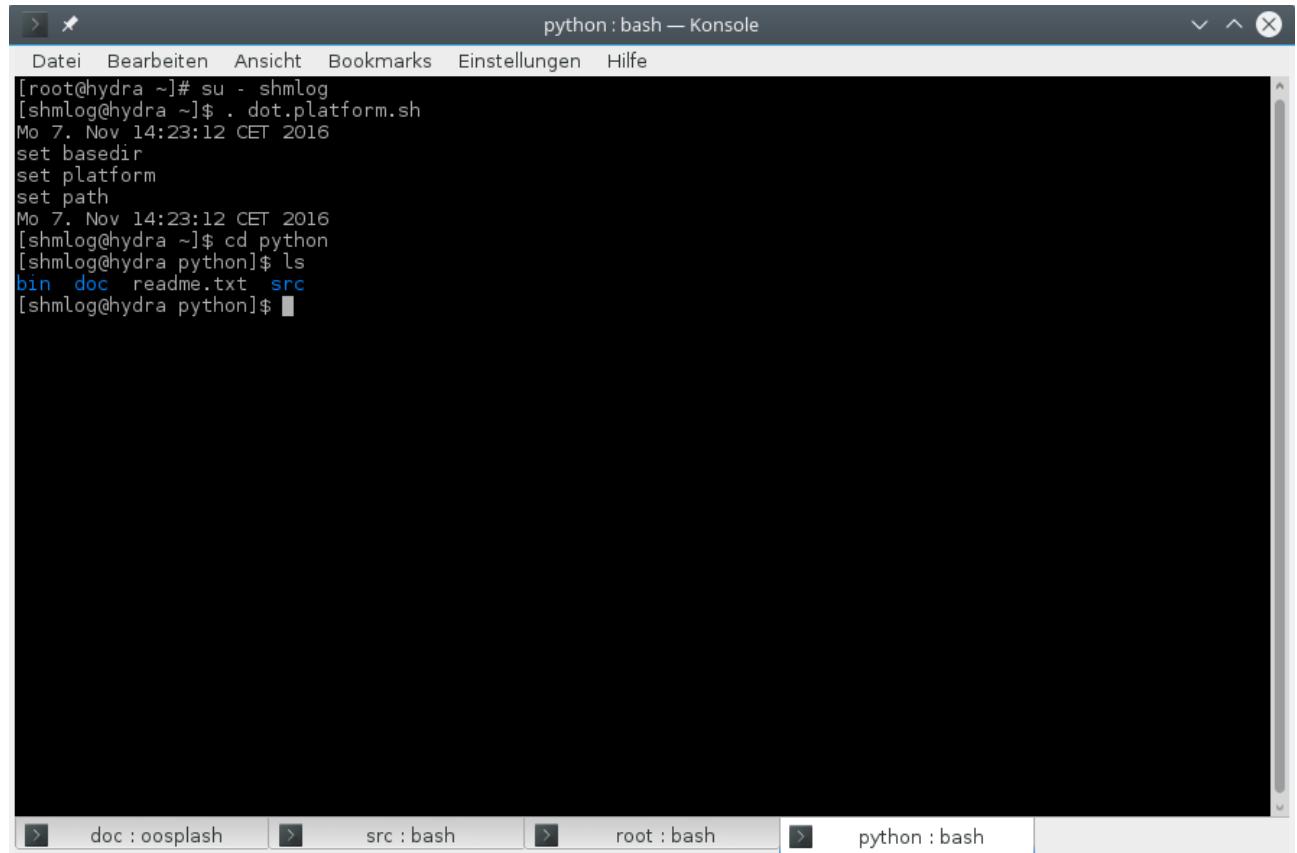
We have in place an implementation that should work out of the box for Linux and python2 and python3.

You have simply to know what you want to build. If you need both its possible to make the first and then copy the thing out of the directory to a safe place, then reset the environment and build the other. No need to build the C module again or to clean up.

## **The python directory**

We start with the basics.

The python stuff is located parallel to src – so we start at BASEDIR/python



A screenshot of a terminal window titled "python : bash — Konsole". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". The terminal content shows a root shell session on a system named "hydra". The user runs "su - shmllog" to switch to a regular user account. Then, ". dot.platform.sh" is run to set environment variables. The user then runs "cd python" to change into the Python source directory. Finally, "ls" is run to list the contents of the directory, which are "bin", "doc", "readme.txt", and "src". The terminal window has tabs at the bottom: "doc : oosplash", "src : bash", "root : bash", and "python : bash" (which is currently active). The background of the terminal window is black, and the text is white.

```
[root@hydra ~]# su - shmllog
[shmllog@hydra ~]$ . dot.platform.sh
Mo 7. Nov 14:23:12 CET 2016
set basedir
set platform
set path
Mo 7. Nov 14:23:12 CET 2016
[shmllog@hydra ~]$ cd python
[shmllog@hydra python]$ ls
bin doc readme.txt src
[shmllog@hydra python]$
```

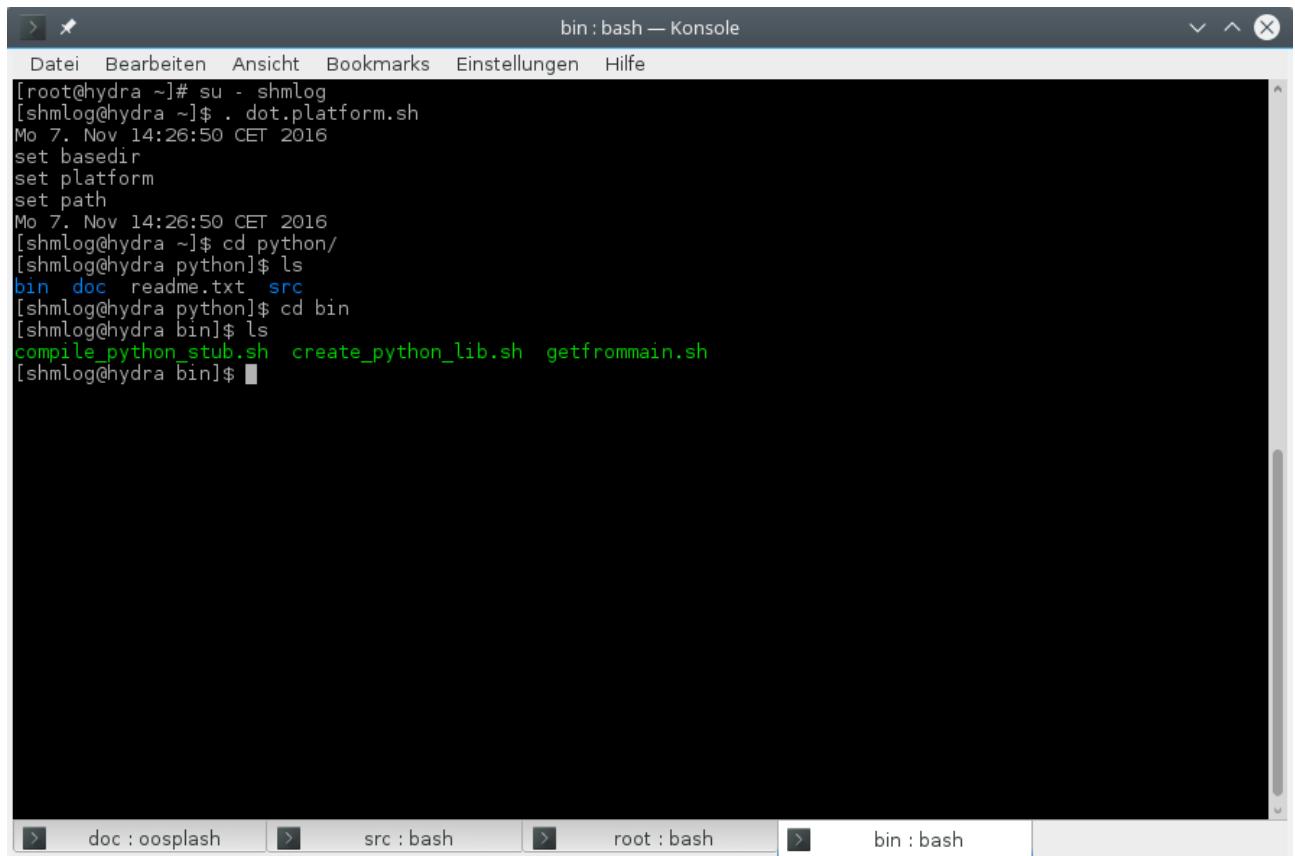
*Illustration 44: The python basedir*

OK. We have a readme – check it – and a bin, a doc and a src.

For the bin its clear, there are some scripts.

## The bin directory

That's my bin so far.



A screenshot of a terminal window titled "bin : bash — Konsole". The window shows a command-line session:

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mo 7. Nov 14:26:50 CET 2016
set basedir
set platform
set path
Mo 7. Nov 14:26:50 CET 2016
[shmlog@hydra ~]$ cd python/
[shmlog@hydra python]$ ls
bin doc readme.txt src
[shmlog@hydra python]$ cd bin
[shmlog@hydra bin]$ ls
compile_python_stub.sh create_python_lib.sh getfrommain.sh
[shmlog@hydra bin]$
```

The terminal has four tabs at the bottom: "doc : oosplash", "src : bash", "root : bash", and "bin : bash", with "bin : bash" being the active tab.

Illustration 45: The python bin directory with the scripts

We have here one script that is interesting now. The rest becomes clear when it comes to a real build cycle. So for now a short info about the scripts

- `compile_python_stub.sh`  
The main build script – its the same for the python layer as the `makeall.sh` for the C module.
- `create_python_lib.sh`  
The helper to compile the bridge C code and link to the library to get – ahem – the library ?  
I think here we have to distinguish the library ( C module ) and the python bridge library for the python stuff. We call the later from now on the python library.
- `getfrommain.sh`  
The script to transfer the headers and the library to python directory's. This is the real thing here, see below.

OK. There were one real thing, but its one for now. The `getfrommain.sh`. It transfers the headers and

the library to the mystery python directory's.

A screenshot of a terminal window titled "python : bash — Konsole". The terminal is running as root on a system named "hydra". The user has run the command ". dot.platform.sh" which sets environment variables like "basedir", "platform", and "path". The user then navigates to the "bin" directory and runs the script "compile\_python\_stub.sh" with arguments "create\_python\_lib.sh" and "getfrommain.sh". The output of this command lists several files: "atrshmlogmodule.c", "atrshmlogtest3.py", "compile\_python\_stub.sh", "dot.python2.sh", "dot.python.sh", "atrshmlogtest2.py", "atrshmlogtest.py", "create\_python\_lib.sh", "dot.python3.sh", and "includes". The terminal has tabs at the bottom: "doc : oosplash", "src : bash", "root : bash", and "python : bash" (which is highlighted).

*Illustration 46: The python source directory*

So that's now the python things, and its the src.

OK. So we transfer with getfrommain.sh from the src what's there to the whole bunch.

Meaning: you should have the real headers in src in place. And the real library. Don't mix platforms or versions – then you have to ignore that script.

If everything is right we can execute the script now. Its mandatory that you are in the python directory for its execution. So we call it relative with bin/getfrommain.sh.

## **Copy headers and lib from the C module**

And so we do it

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mo 7. Nov 14:33:33 CET 2016
set basedir
set platform
set path
Mo 7. Nov 14:33:33 CET 2016
[shmlog@hydra ~]$ cd python
[shmlog@hydra python]$ ls
bin doc readme.txt src
[shmlog@hydra python]$ cd bin
[shmlog@hydra bin]$ ls
compile_python_stub.sh create_python_lib.sh getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra python]$ ls src
atrshmlogmodule.c atrshmlogtest3.py compile_python_stub.sh dot.python2.sh dot.python.sh
atrshmlogtest2.py atrshmlogtest.py create_python_lib.sh dot.python3.sh includes
[shmlog@hydra python]$ bin/getfrommain.sh
[shmlog@hydra python]$
```

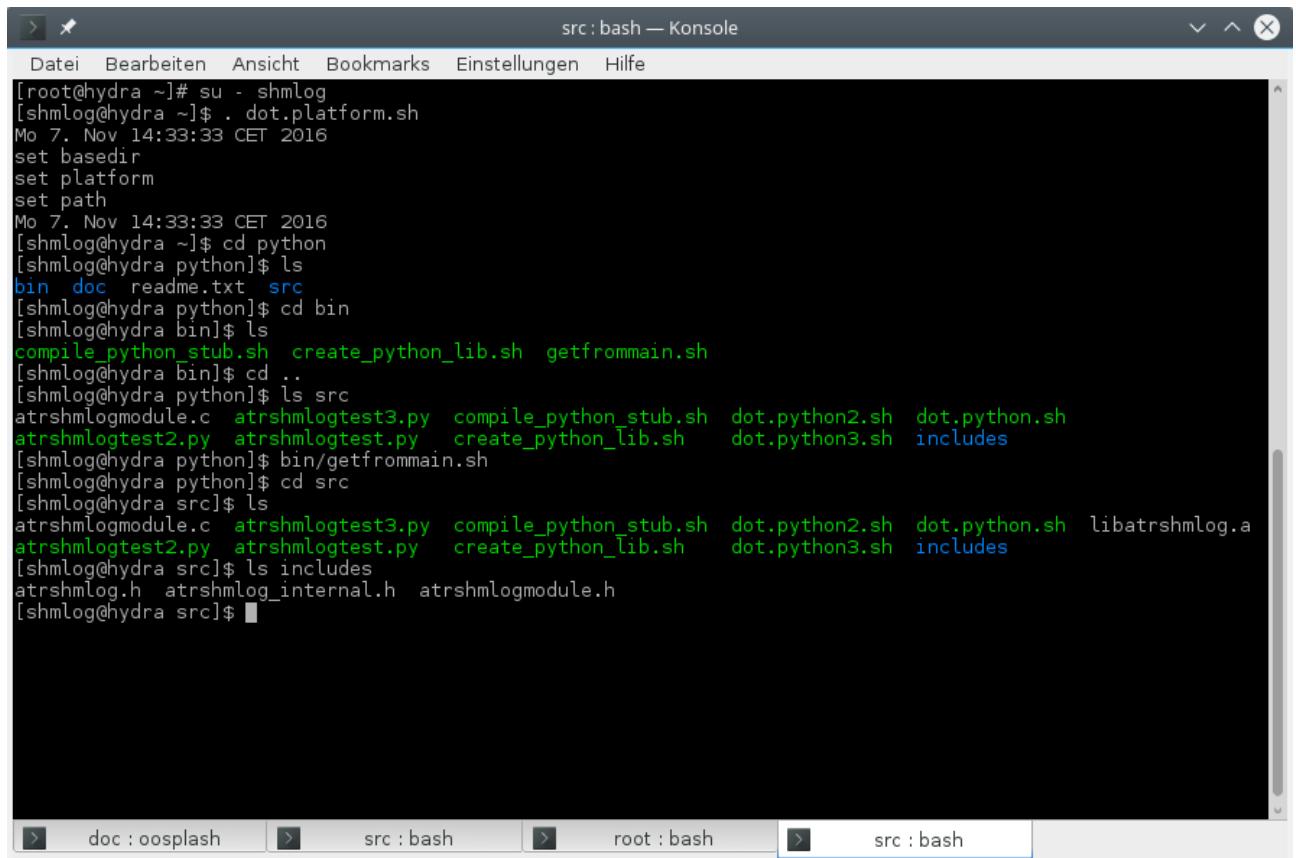
*Illustration 47: Transfer of lib and headers before build*

Not much noise. Well, we will see....

Now its time to check for the src directory's.

## **Change into your src directory**

We change and then see for it



The screenshot shows a terminal window titled "src : bash — Konsole". The terminal is running as root on a system named "hydra". The user has navigated to the source directory ("src") and listed its contents. The output of the "ls" command shows several files and scripts:

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mo 7. Nov 14:33:33 CET 2016
set basedir
set platform
set path
Mo 7. Nov 14:33:33 CET 2016
[shmlog@hydra ~]$ cd python
[shmlog@hydra python]$ ls
bin doc readme.txt src
[shmlog@hydra python]$ cd bin
[shmlog@hydra bin]$ ls
compile_python_stub.sh create_python_lib.sh getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra python]$ ls src
atrshmlogmodule.c atrshmlogtest3.py compile_python_stub.sh dot.python2.sh dot.python.sh
atrshmlogtest2.py atrshmlogtest.py create_python_lib.sh dot.python3.sh includes
[shmlog@hydra python]$ bin/getfrommain.sh
[shmlog@hydra python]$ cd src
[shmlog@hydra src]$ ls
atrshmlogmodule.c atrshmlogtest3.py compile_python_stub.sh dot.python2.sh dot.python.sh libatrshmlog.a
atrshmlogtest2.py atrshmlogtest.py create_python_lib.sh dot.python3.sh includes
[shmlog@hydra src]$ ls includes
atrshmlog.h atrshmlog_internal.h atrshmlogmodule.h
[shmlog@hydra src]$
```

The terminal window has tabs at the bottom: "doc : oosplash", "src : bash", "root : bash", and "src : bash". The "src : bash" tab is currently active.

*Illustration 48: Inside the source directory ready for build*

The src is the real thing. We have again the – adapted - scripts here, so we are free to change them if we need. For reference there are the scripts in bin, so don't change them in bin till you have done the whole thing. Start only in src with changing.

We switch to src and now comes the list of the files in there.

- `compile_python_stub.sh`  
As already said. The main build script..
- `create_python_lib.sh`  
The helper to compile the bridge and link.
- `dot.python.sh`  
The setting of the environment variables. Next thing to do.
- `atrshmlogmodule.c`

The python bridge code.

- dot.python2.sh

The setting of the environment variables. Its the python2 this time for Linux.

- dot.python3.sh

The setting of the environment variables. Its the python3 this time for Linux. So you can change the dot.python.sh and have it still here for reference.

- libatrshmlog.a

The copy of the library from the C module (check time stamp and size and whatever you need too ... should be from BASEDIR/src )

- atrshmlogtest.py

Start the simple test.

- atrshmlogtest2.py

Helper to start the simple test with python2.

- atrshmlogtest3.py

Helper to start the simple test with python3 – so you can change the atrshmlogtest.py and still have it here as a reference.

For the directory's : includes is clear – check for the files and for the time stamp and length of the headers from BASEDIR/src.

OK. Now that we know that tree, check for the others.

When you are ready, you can read on.

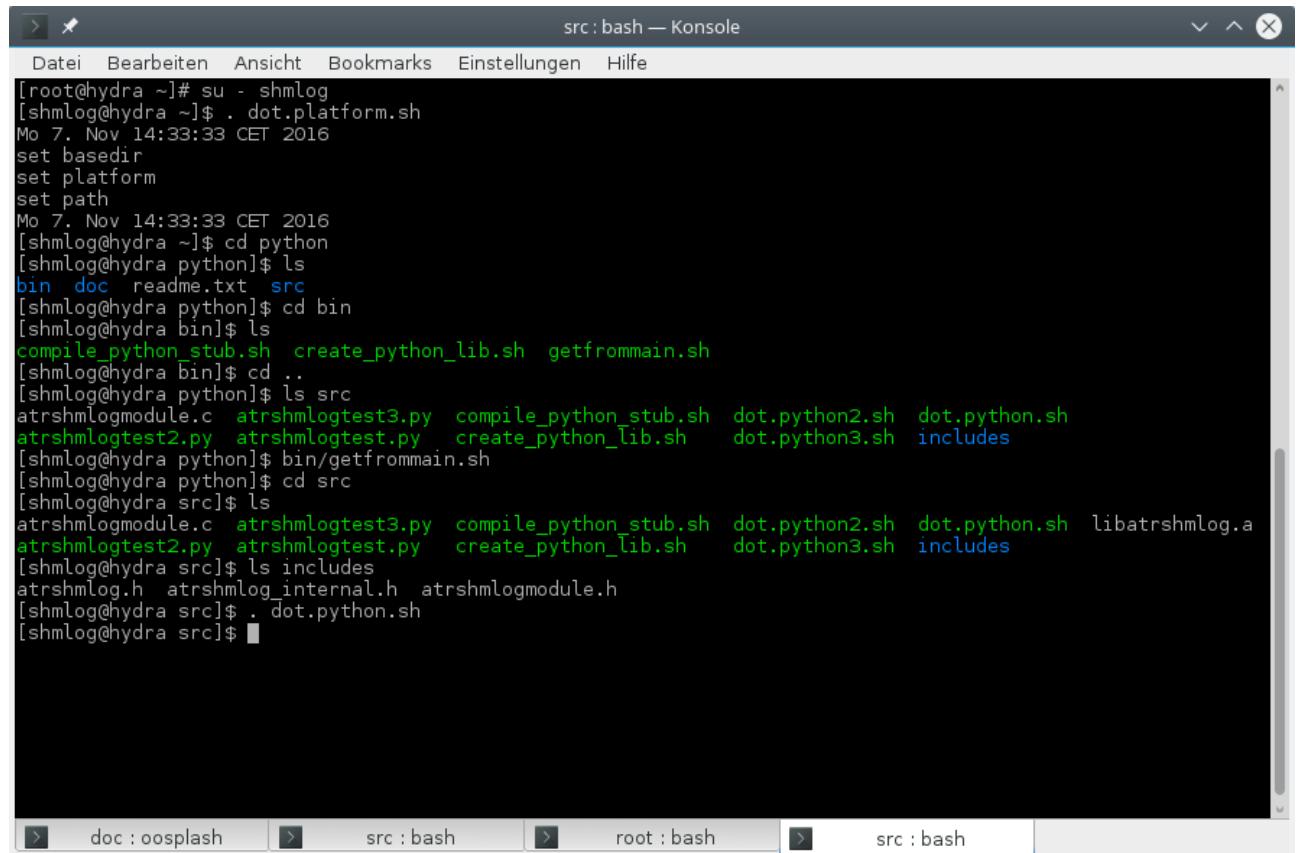
Ready ? So fast – well, if you said so.

Its now time to do the thing.

## **Setting the environment**

We have to put the platform in place first. So if you have not done it switch to BASEDIR and source the right dot file.

We then simply source dot.python.sh



The screenshot shows a terminal window titled "src : bash — Konsole". The window contains a command-line session where the user is setting up a build environment. The session starts with the user switching to the root account and running a script to set the platform. This is followed by navigating to the Python directory, listing files, and then navigating into the "bin" directory. The user runs several shell scripts to compile Python stubs and create Python library files. Finally, they navigate to the "src" directory and list its contents, which include various Python files and header files. The terminal window has a standard Linux-style interface with tabs at the bottom labeled "doc : oosplash", "src : bash", "root : bash", and "src : bash".

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mo 7. Nov 14:33:33 CET 2016
set basedir
set platform
set path
Mo 7. Nov 14:33:33 CET 2016
[shmlog@hydra ~]$ cd python
[shmlog@hydra python]$ ls
bin doc readme.txt src
[shmlog@hydra python]$ cd bin
[shmlog@hydra bin]$ ls
compile_python_stub.sh create_python_lib.sh getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra python]$ ls src
atrshmlogmodule.c atrshmlogtest3.py compile_python_stub.sh dot.python2.sh dot.python.sh
atrshmlogtest2.py atrshmlogtest.py create_python_lib.sh dot.python3.sh includes
[shmlog@hydra python]$ bin/getfrommain.sh
[shmlog@hydra python]$ cd src
[shmlog@hydra src]$ ls
atrshmlogmodule.c atrshmlogtest3.py compile_python_stub.sh dot.python2.sh dot.python.sh libatrshmlog.a
atrshmlogtest2.py atrshmlogtest.py create_python_lib.sh dot.python3.sh includes
[shmlog@hydra src]$ ls includes
atrshmlog.h atrshmlog_internal.h atrshmlogmodule.h
[shmlog@hydra src]$ . dot.python.sh
[shmlog@hydra src]$
```

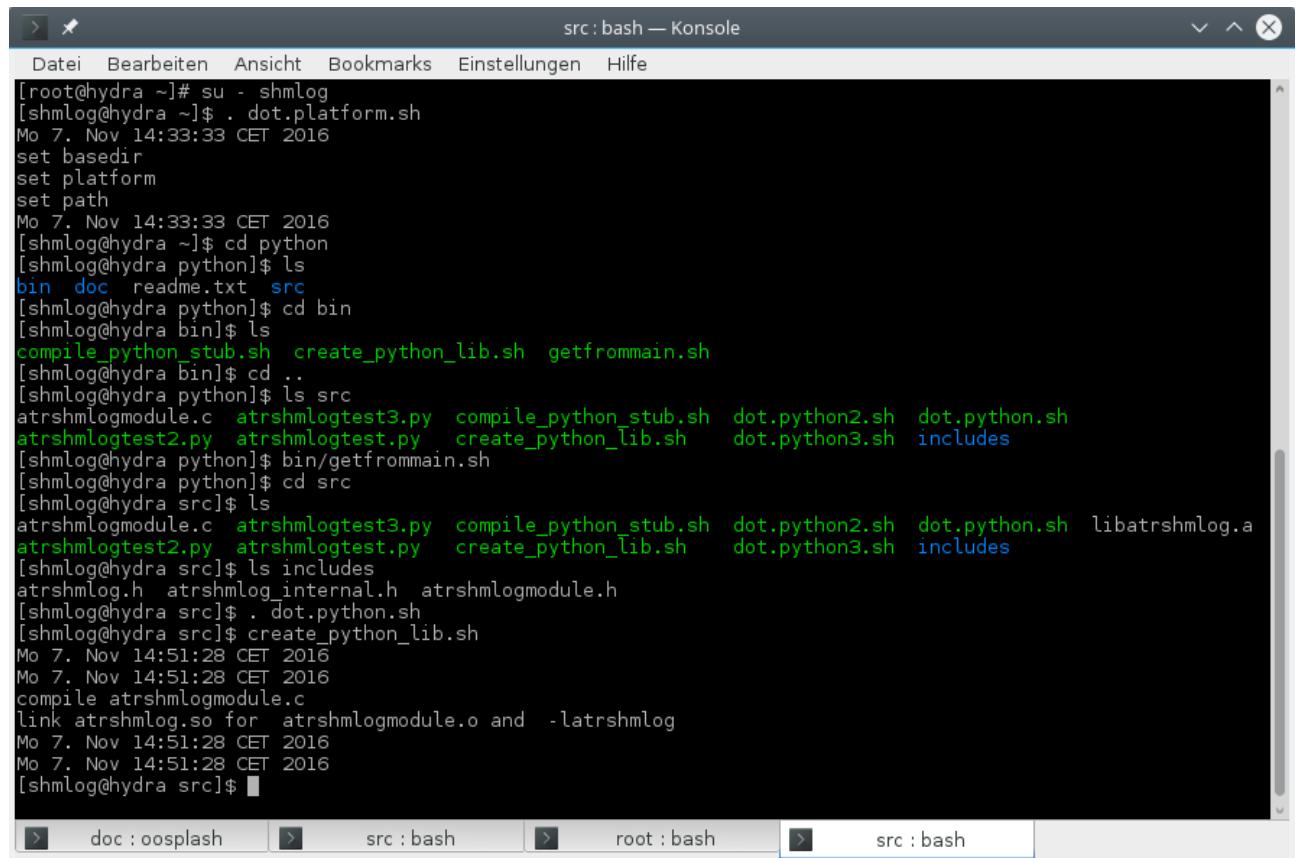
*Illustration 49: Setting the build environment*

No noise here. If you insist you can check the environment.

Next is to start the build.

## ***Building with create\_python\_lib.sh***

We start the script



The screenshot shows a terminal window titled "src : bash — Konsole". The terminal is running a series of commands to build a Python library. The commands include navigating to the source directory, running setup scripts, and compiling source code. The output shows the creation of Python modules and header files, along with timestamp information.

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mo 7. Nov 14:33:33 CET 2016
set basedir
set platform
set path
Mo 7. Nov 14:33:33 CET 2016
[shmlog@hydra ~]$ cd python
[shmlog@hydra python]$ ls
bin doc readme.txt src
[shmlog@hydra python]$ cd bin
[shmlog@hydra bin]$ ls
compile_python_stub.sh create_python_lib.sh getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra python]$ ls src
atrshmlogmodule.c atrshmlogtest3.py compile_python_stub.sh dot.python2.sh dot.python.sh
atrshmlogtest2.py atrshmlogtest.py create_python_lib.sh dot.python3.sh includes
[shmlog@hydra python]$ bin/getfrommain.sh
[shmlog@hydra python]$ cd src
[shmlog@hydra src]$ ls
atrshmlogmodule.c atrshmlogtest3.py compile_python_stub.sh dot.python2.sh dot.python.sh libatrshmlog.a
atrshmlogtest2.py atrshmlogtest.py create_python_lib.sh dot.python3.sh includes
[shmlog@hydra src]$ ls includes
atrshmlog.h atrshmlog_internal.h atrshmlogmodule.h
[shmlog@hydra src]$ . dot.python.sh
[shmlog@hydra src]$ create_python_lib.sh
Mo 7. Nov 14:51:28 CET 2016
Mo 7. Nov 14:51:28 CET 2016
compile atrshmlogmodule.c
link atrshmlog.so for atrshmlogmodule.o and -latrshmlog
Mo 7. Nov 14:51:28 CET 2016
Mo 7. Nov 14:51:28 CET 2016
[shmlog@hydra src]$
```

*Illustration 50: Create the python library*

And there we are.

One compile and one link line.

All what's left is the test.

## Testing the python bridge

We have first to create a shared memory buffer with atrshmlogcreate. I will use 4711 and 8 for it. Then the init for the area. Next the test of the bridge. Then the reader and the convert.

At last we can check for the result log.

And because it was already done for the C test program we do it in short form now.



```
[shmlog@hydra src]$ cd
[shmlog@hydra ~]$ cd src
[shmlog@hydra src]$ atrshmlogcreate 4711 8
shm log create.
logsystem version is 1.
ipckey is 0X1267
count of buffer is 8

paste this into the user process environment startshell
or into the profile or ENV file

ATRSHMLOG_ID="7503884"
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
[shmlog@hydra src]$ . dot.atrshmlog
[shmlog@hydra src]$ atrshmloginit
shm log attach and init.
logsystem version is 1.
[shmlog@hydra src]$ cd ..
[shmlog@hydra ~]$ cd python
[shmlog@hydra python]$ cd src
[shmlog@hydra src]$ atrshmlogtest.py
Traceback (most recent call last):
  File "./atrshmlogtest.py", line 17, in <module>
    greeting = sys.argv[1]
IndexError: list index out of range
[shmlog@hydra src]$ atrshmlogtest.py hallo
[shmlog@hydra src]$ cd
[shmlog@hydra ~]$ cd src
[shmlog@hydra src]$ atrshmlogstopreader
shm log attach and set reader flag and pid.
logsystem version is 1.
pid before 0
flag before 0
[shmlog@hydra src]$ atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
logging done.
writer data : time 1206300 count 1 per use 1206300
[shmlog@hydra src]$ atrshmlogconv d1
shm log converter from file 'd1/0/atrshmlog_p7115_t7115_s0_f0.bin' to file 'd1/0/atrshmlog_p7115_t7115_s0_f0.txt'
id 1 acquiretime      530 pid    7115 tid     7115 slavetime      1150 readertime     8900 payloadsize      34
shmbuffer 0 filenumber 0 sequence 0
[shmlog@hydra src]$ cat d1/0/atrshmlog_p7115_t7115_s0_f0.txt
0000007115 0000000000007115 000 0000000000000000 000051139880055170 000051139880061550 0000000000000006380 14785
27029515082576 1478527029515084900 00000000000000002324 0000000001 I 0000000042 hallo
[shmlog@hydra src]$
```

Illustration 51: Test of the python library

HM. One glitch, but a typo is OK at least, the rest worked as expected.

## **Details**

Now we make it for the bridge in C.

Take your favorite text editor and open the atrshmlogmodule.c.

We start with the usual comments about what it is...

Then we enter the python version code thing. Its set via -D in the build script. And its value is given from the dot file.

So here we have a first check group. Have we made a new version in python we will need a new check here ...

Next we check our settings in the include directory we have given the thing to use.

Again a wrong or new version would be found here.

After the checks we set a define for the size\_t thing and include the Python.h and our module and the C module files.

Now we have a small helper, a struct thing for conversions. We use large numbers to transport addresses of the shared memory area around.

Then we have the error object. I use it in case the supported parameters for a call are wrong. For the module calls I return that returncode – no exceptions here. So if you get an exception its normally a problem with the supported parameters. For internal error you have to see what's the returncode.

Most functions even do not have one, they are simple getter's. So don't overdue the checking....

Next a small helper for making a list.

OK. And now we start the real thing.

The core functions.

After I found the way to implement the C API capsule I made them. They simply call the C module functions. And they make on low level C the conversion from the internal types to the things I use in the bridge code.

So the rough hundred functions that follow are the real working code functions.

They are the members in the C API pointer buffer.

And they are used in the interface code as the function that does the thing. So you get the same when you call them in the interpreter – via the interface function – or direct via the C API header trick.

Simple first one is attach.

For gettime I decided not to check events – its not worth that in a layer of multiple functions and with check for parameters in the way python does it – so its always on.

The write0 and write. Didn't check for binary now, according to docu it could work...

Again simple stuff for sleep nanos, and so long till we hit get\_inittime.

Here I have the version in C with giving back two values via pointers. So we do not use the struct as in the C module here.

Again simple stuff till we hit get\_realtime. Same goes here for the use of pointers as for get\_inittime. We do not use the struct.

Again simple stuff till we hit the two we normally don't need in a client, the read and read\_fetch.

Both can be used, but only read\_fetch is the actual one. The read will disappear in the next version. For now we have both.

The core function simply makes again the type conversions into the types I use in the python part. And again the inittime and he lasttime are split into two pointer assignment operations.

And again that boring simple core functions.

Last are the two that I don't need in the C world, the things to inspect and change the area by a python user, the peek and poke things – same as for java I try to give you the minimal support and that's it.

So rough 800 and about 20 percent of the thing is done.

Now the interface code follows.

I had first the actual call to the C module functions in here, but after the core functions were introduced I had also to switch to them – its perhaps a bit slower that way. I will make some tests in the next version and eventually then support gettime and write direct here, and you will not have the same in the C API, but it will work still the same way in the C module.

Again the first is the attach. And its a simple one.

So we now have simple getter like calls to the core functions. No big deal.

The write is the first to use parameters. And no, I didn't try the keywords. Would be nice but also a bit slower I guess. If anyone is interested I will add them too. Simply sent me a post card (with a credit card attached ) and you are in ....

Next interesting stop is get\_statistics. I use here the helper and create the result list.

Next some string things. See for the python 2 and 3 way to make strings if you have a problem with it.

Again simple boring getter like and setter like stuff till we hit get\_time.

This time we need as result a tuple, so I do here the conversion from the given values to a tuple.

If you find this interesting – I can't help you ...

Again simple setter and getter stuff. Then we hit get\_realtime

Again the pointer values are turned into a tuple.

Now we have from time to time the area and the thread locals in – both addresses and so pointers in C – and in the python interpreter I use a number – so I do the conversion in a bit unusual way by the union here – no cast or assignment to another type. So its up to the layout of the platform what you get as a number – no way for me to tell it – but its consistent that way.

OK. Now the big two again – read and read\_fetch.

I have tried to take the buffer stuff out of reach if you don't use them – its a half MB after all – and moved the thing to thread local if you really plan to do it in multithread program way.

For the most things I have simple checks – and again for the parameters an exception if it fails. The rest is simple. I construct a tuple and you get what I have.

A tuple with one returncode – bad thing happened.

A tuple with a code and a length only – empty buffer found, rest OK.

For the real thing you get near 40 values back, and its the buffer as a byte array. So you are in with the binary things here.

After everything goes well you can use the tuple in the way you want. And I do not give back the buffer then – its assumed you have no read in a client, only in a dedicated reader – so you have to accept it will reuse the buffer from now on.

Next is again the simple stuff of getter and setter like functions. Starts with verify.

So we reach the end of the interface functions round about 3950 with peek.

Now we have the methods array.

If there is more time I will start better comment strings for the methods. See next version.

About 300 lines later we are in a little helper to clean the init function. I use here a helper to make the C API array – that's not a real need, could have made an initialization instead like in the C Module for the static buffers. But this is the way of the documentation for python org, so I didn't make it different.

After the hundred are in, we reach the python 3 stuff.

The module is described with the module struct here.

To make things a bit cleaner in the init function I use some defines.

A return code helper – the python 2 is void, python3 delivers a reference – and an name adjustment for the one external function.

OK. This wasn't hard at all.

What's left is the init function itself.

I try some runtime checks here. Its normally too late for it – the different flavors are simply not link compatible for the used string and bytes buffer stuff. But if you ever get hands on a next version that is – here is the safety against wrong use.

Next comes the create for the 3 and initmodule for 2.

Well, could have made a helper to make them identical – but then I would have needed some tricks with defaults and this seemed not right to me here. So we have to live with the #if here.

Setting up the error object is same.

Also the helper for the core functions in the API array.

So next is the capsule and we are in for the module.

Wait – forgot the constants...

Having such things in C or Java it felt right to me to make them in python too.

So we have here the helper for the write0 and write. OK, perhaps I overdid it – but I used one and it worked.

So much for the details.

Now check the test scripts and you should be able to start the new logging thing.

## The perl language support

After I had done the initial jni stuff it was clear for me to make it to other languages too.

So on my first list I had python, perl and for those others SWIG in case they use that.

For the perl I had to start at with my python experience.

I know something after 20 years of use for perl, but no internals so far. Only from the panther book the things how it should work with XS and with SWIG.

So I started with XS. It is after all the default for all modules for perl distros itself.

But I ran in some silly problems. Didn't get the right answers for my questions. Began to become a little bit frustrated after the easy way that python had made it.

So I switched to SWIG.

OK, that's not too hard. For the first test drive it took 30 minutes and I was back in business.

See Attrshmlog.i for it.

Then the small adjustments for names and that stuff, the ignore of not needed or pollution of name space things.

The remaining problems were the missing gettime – OK, that was easy, use an additional C source for the thing and make it there.

The return for the get\_inittime and get\_realtime came next. First found a thing about making a reference pointer work with some – sorry to say it – ugly helper functions like alloc, fetch, store and delete.

But it looked like it could work.

Checked some other hits from google and - BINGO.

With the OUTPUT things got back on the right direction again.

Same problem, only bigger in code was the get\_statistics. Again a helper wrapper with OUTPUT solved it. So we are now in for 100 values – from which for now 85 are in use. The rest is 0.

Took then 3 hours to make it from the python core functions to the perl wrapper code for read and read\_fetch.

But I skipped one thing.

Had core dumps when I first integrated the things and loaded the module.

Always at end I encountered a core.

I installed the debug stuff and then made it with gdb to the point of wreck.

Found a free of an environ[i] on the stack.

Environ ? Free ?

Then something made click and I checked the attach code. And yes, I had some putenv stuff in. Don't know why perl insists to free the environment strings, but it does it. And so I made a small hack in the internal header. And a #if in putenv in the attach.

If you really NEED the putenv – which is for a login shell only – you now have to switch that define to 1 in the build of attach. So you have now normally no putenv in attach.

And it worked.

So after I had made the perl module I got also two test scripts, one for the usual small test and the other for the read\_fetch. Nice to have it, but I doubt it will be of help for the reader stuff, more for debugging.

## Perl XS

This was strange. I did some things – and they worked. Others didn't.

For example I could make the getter's for the int values. OK. But the getter's for 64 bit simply didn't work right.

So I had also the core dump thing – which I could solve in the SWIG session.

All in all this means you have for now to install SWIG and then can build it.

If you have problems to do that contact me and I will give the XS a second try. Or help you with the SWIG.

## How it works

The SWIG uses its definition file to build a wrapper C code file and the perl module pm file for the use directive. So you have in place the swig wrapper, the additional C code file with the helpers and the C module itself.

The module is a direct used implementation. Under cover you use the C module. The module is capable of doing its job simply by using the internal infrastructure as in a C environment. No need for changes inside.

This means you get in theory the same speed as for the C module itself. The only thing that slows down is the perl parameter handling in the bridge code itself.

The rest is more or less simple. Use of the function and no object stuff. Simple raw data types, no classes, no exceptions and no use of the perl side. Barely some string creations.

If you are interested the atrshmlog\_perlwrapper.c is a good start.

The module gives also the a small number of constants to the user.

Also the enum's are there, but here you have to use the C notation with the in C usual prefix.

A simple use gets it in.

## **How to use it**

The user needs the following parts:

- Shared library or dll Attrshmlog

This is the bridge code from the SWIG code atrshmlog\_wrap.c , the atrshmlog\_perlwrapper.c and the linked library from the C module, libatrshmlog.a. You need also the compiler support lib's and in case of the mingw port the additional dll 's of the mingw system.

- The support programs

You need the matching C support programs for the basic stuff you don't want to reinvent. Reader, converter, even create and delete. So if you need a pure (?) perl only solution you can replace them with your own code perl counterparts. But for now I deliver nothing so you should start with the C programs.

So you first have to build it – again I am not delivering binary code, only source, so if you are interested in binary you have to contact me and I will see what I can do. But for now you have to start with the perl bridge file atrshmlog\_perlwrapper.c and Attrshmlog.i.

## **How to build it in the first place**

We start with the C module.

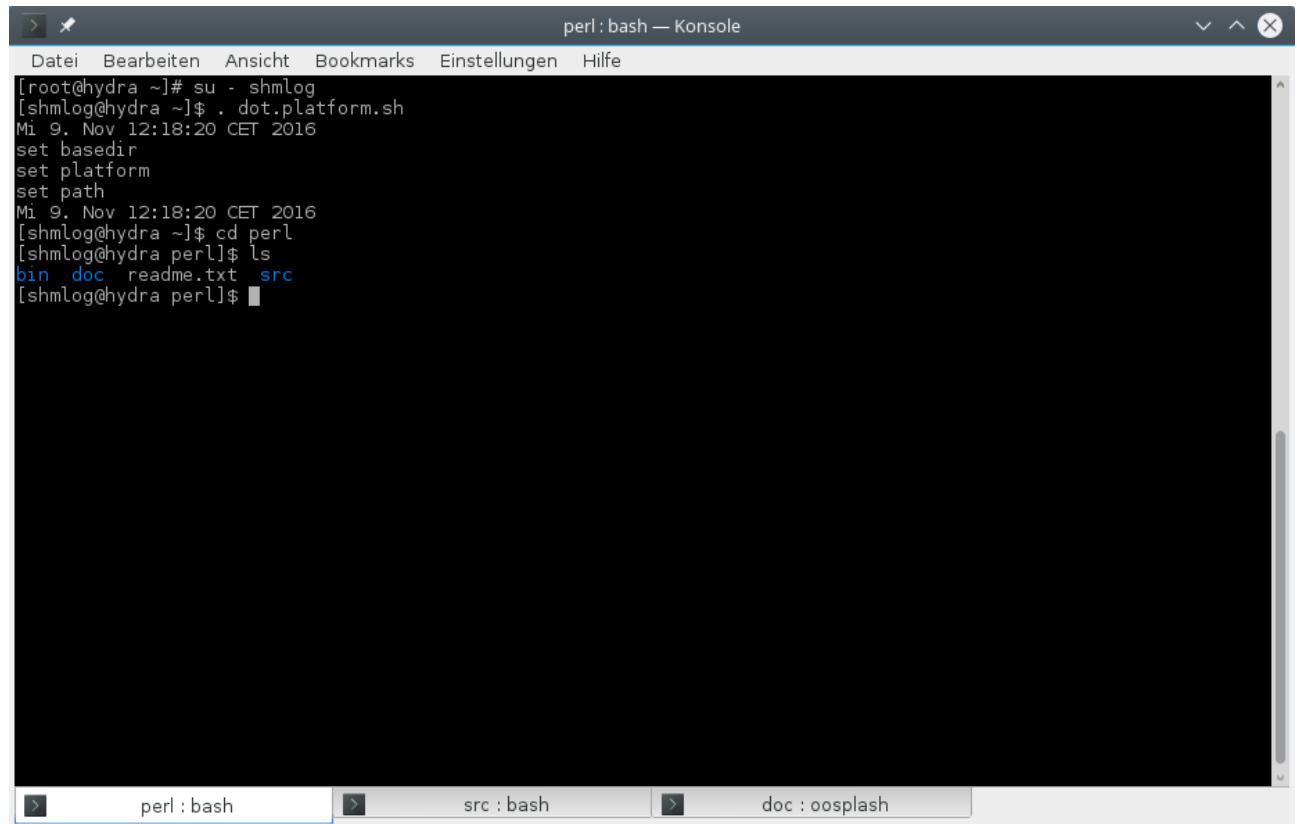
After this is in place we have to check our next options.

We have in place an implementation that should work out of the box for Linux and perl 5. Didn't test perl 6 yet, so no perl 6 for now from me – but for SWIG I think it should work too.

## **The perl directory**

We start with the basics.

The perl stuff is located parallel to src – so we start at BASEDIR/perl



A screenshot of a terminal window titled "perl : bash — Konsole". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". The terminal content shows a shell session:

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mi 9. Nov 12:18:20 CET 2016
set basedir
set platform
set path
Mi 9. Nov 12:18:20 CET 2016
[shmlog@hydra ~]$ cd perl
[shmlog@hydra perl]$ ls
bin doc readme.txt src
[shmlog@hydra perl]$
```

The terminal has three tabs at the bottom: "perl : bash" (selected), "src : bash", and "doc : oosplash".

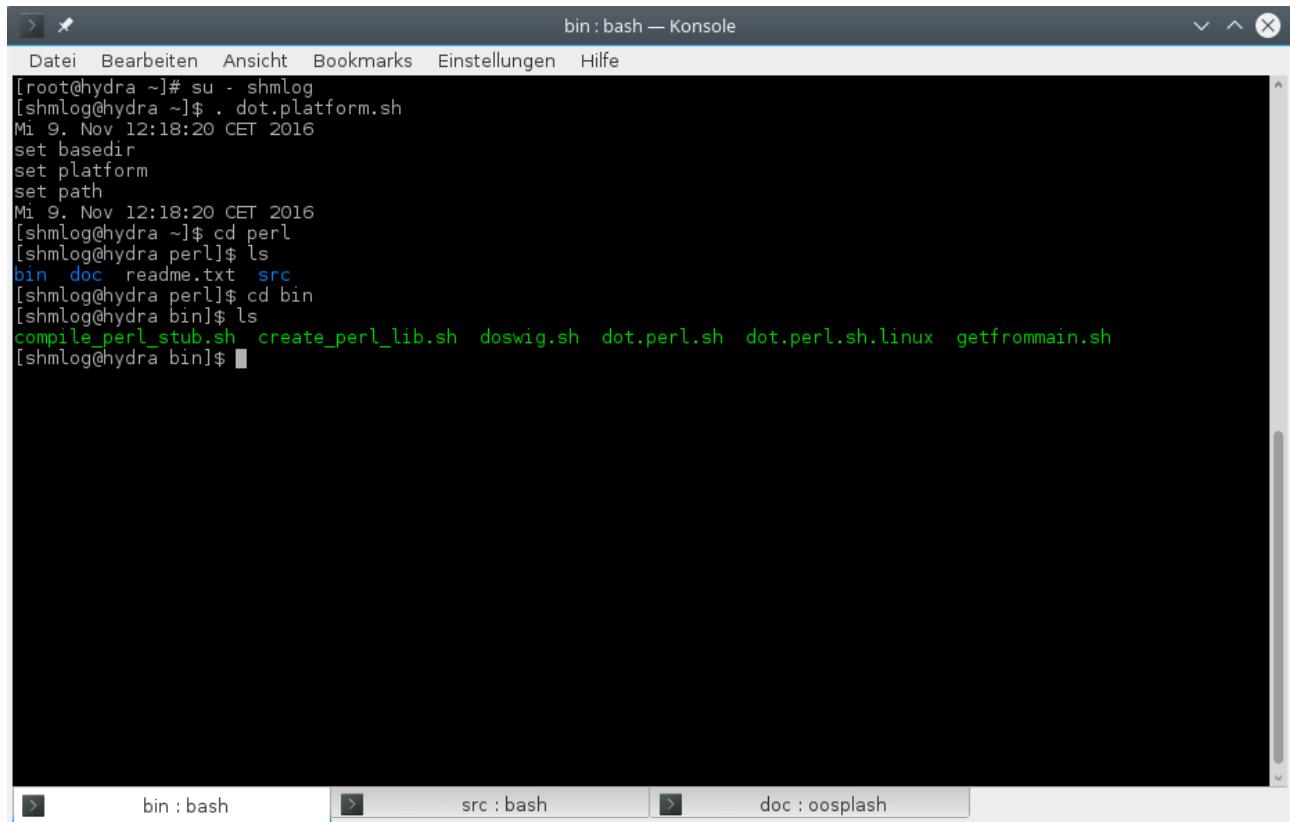
*Illustration 52: The perl basedir*

OK. We have a readme – check it – and a bin, a doc and a src.

For the bin its clear, there are some scripts.

## The bin directory

That's my bin so far.



A screenshot of a terminal window titled "bin : bash — Konsole". The window shows a command-line session where the user has navigated to the "bin" directory and listed its contents. The output of the "ls" command is as follows:

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mi 9. Nov 12:18:20 CET 2016
set basedir
set platform
set path
Mi 9. Nov 12:18:20 CET 2016
[shmlog@hydra ~]$ cd perl
[shmlog@hydra perl]$ ls
bin doc readme.txt src
[shmlog@hydra perl]$ cd bin
[shmlog@hydra bin]$ ls
compile_perl_stub.sh create_perl_lib.sh doswig.sh dot.perl.sh dot.perl.sh.linux getfrommain.sh
[shmlog@hydra bin]$
```

The terminal window has tabs at the bottom labeled "bin : bash", "src : bash", and "doc : oosplash".

Illustration 53: The bin directory with the scripts

We have here one script that is interesting now. The rest becomes clear when it comes to a real build cycle. So for now a short info about the scripts

- `compile_perl_stub.sh`  
The main build script – its the same for the perl layer as the `makeall.sh` for the C module.
- `create_perl_lib.sh`  
The helper to compile the bridge C code and link to the library to get – ahem – the library ?  
I think here we have to distinguish the library ( C module ) and the perl bridge library for the perl stuff. We call the later from now on the perl library.
- `getfrommain.sh`  
The script to transfer the headers and the library to perl directory's. This is the real thing here, see below.

- doswig.sh

This is the helper to do the swig generation for the SWIG wrapper and the perl pm file.

- dot.perl.sh

The environment setter.

- dot.perl.sh.linux

The Linux version – for now the same.

OK. There were one real thing, but its one for now. The getfrommain.sh. It transfers the headers and the library to the mystery perl directory's.

```

Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mi 9. Nov 12:31:10 CET 2016
set basedir
set platform
set path
Mi 9. Nov 12:31:10 CET 2016
[shmlog@hydra ~]$ cd perl
[shmlog@hydra perl]$ ls
bin doc readme.txt src
[shmlog@hydra perl]$ cd bin
[shmlog@hydra bin]$ ls
compile_perl_stub.sh create_perl_lib.sh doswig.sh dot.perl.sh dot.perl.sh.linux getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra perl]$ ls src
Atrshmlog.i atrshmlogread.pl compile_perl_stub.sh doswig.sh dot.perl.sh.linux
atrshmlog_perlwrapper.c atrshmlogtest.pl create_perl_lib.sh dot.perl.sh includes
[shmlog@hydra perl]$ 
```

*Illustration 54: The perl source directory*

So that's now the perl things, and its the src.

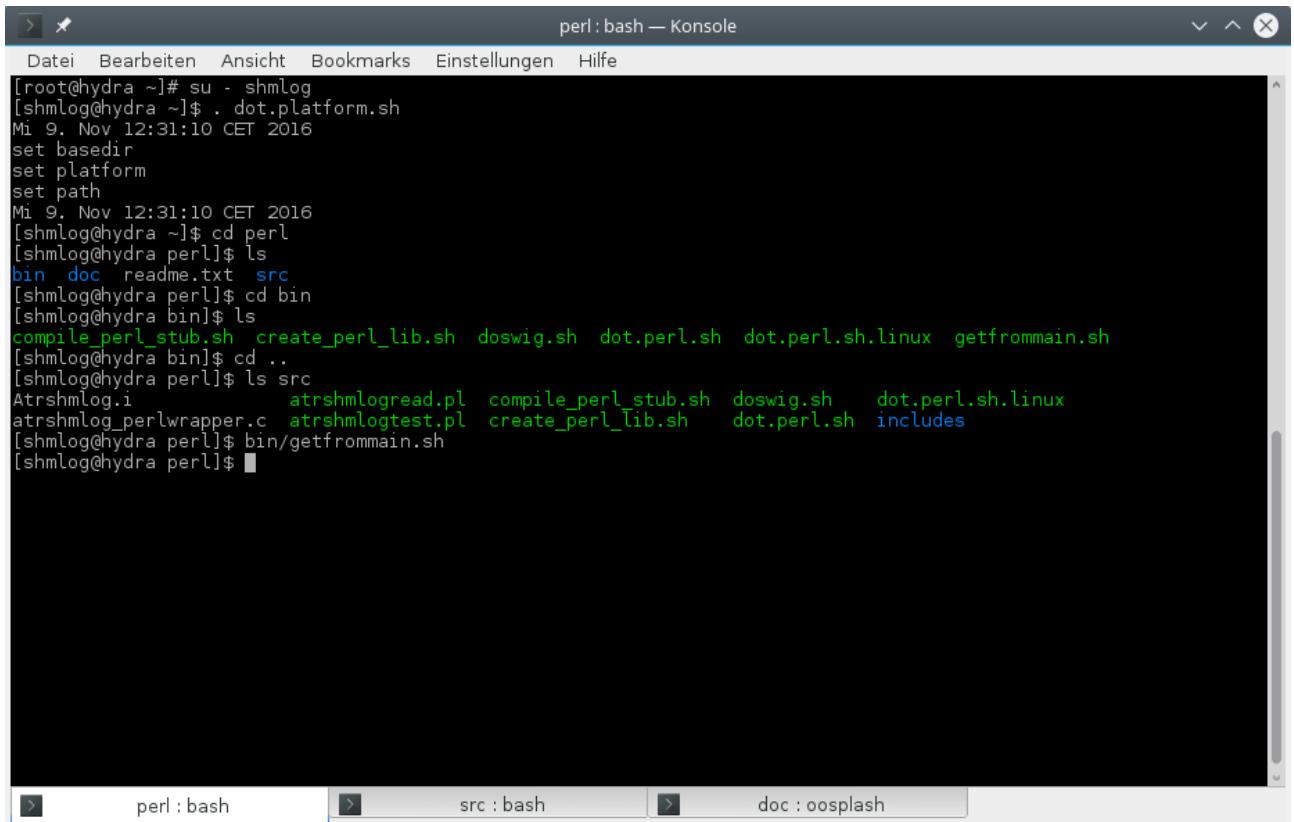
OK. So we transfer with getfrommain.sh from the src what's there to the whole bunch.

Meaning: you should have the real headers in src in place. And the real library. Don't mix platforms or versions – then you have to ignore that script.

If everything is right we can execute the script now. Its mandatory that you are in the perl directory for its execution. So we call it relative with bin/getfrommain.sh.

## **Copy headers and lib from the C module**

And so we do it



The screenshot shows a terminal window titled "perl : bash — Konsole". The window contains a command-line session where a user is navigating through directory structures and running scripts to copy files. The session starts with the user switching to the root account and running a setup script. It then moves into the perl directory, lists files, and runs a script to compile a Perl stub. Finally, it lists files again, showing the transferred files.

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mi 9. Nov 12:31:10 CET 2016
set basedir
set platform
set path
Mi 9. Nov 12:31:10 CET 2016
[shmlog@hydra ~]$ cd perl
[shmlog@hydra perl]$ ls
bin doc readme.txt src
[shmlog@hydra perl]$ cd bin
[shmlog@hydra bin]$ ls
compile_perl_stub.sh create_perl_lib.sh doswig.sh dot.perl.sh dot.perl.sh.linux getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra perl]$ ls src
Atrshmlog.i atrshmlogread.pl compile_perl_stub.sh doswig.sh dot.perl.sh.linux
atrshmlog_perlwrapper.c atrshmlogtest.pl create_perl_lib.sh dot.perl.sh includes
[shmlog@hydra perl]$ bin/getfrommain.sh
[shmlog@hydra perl]$
```

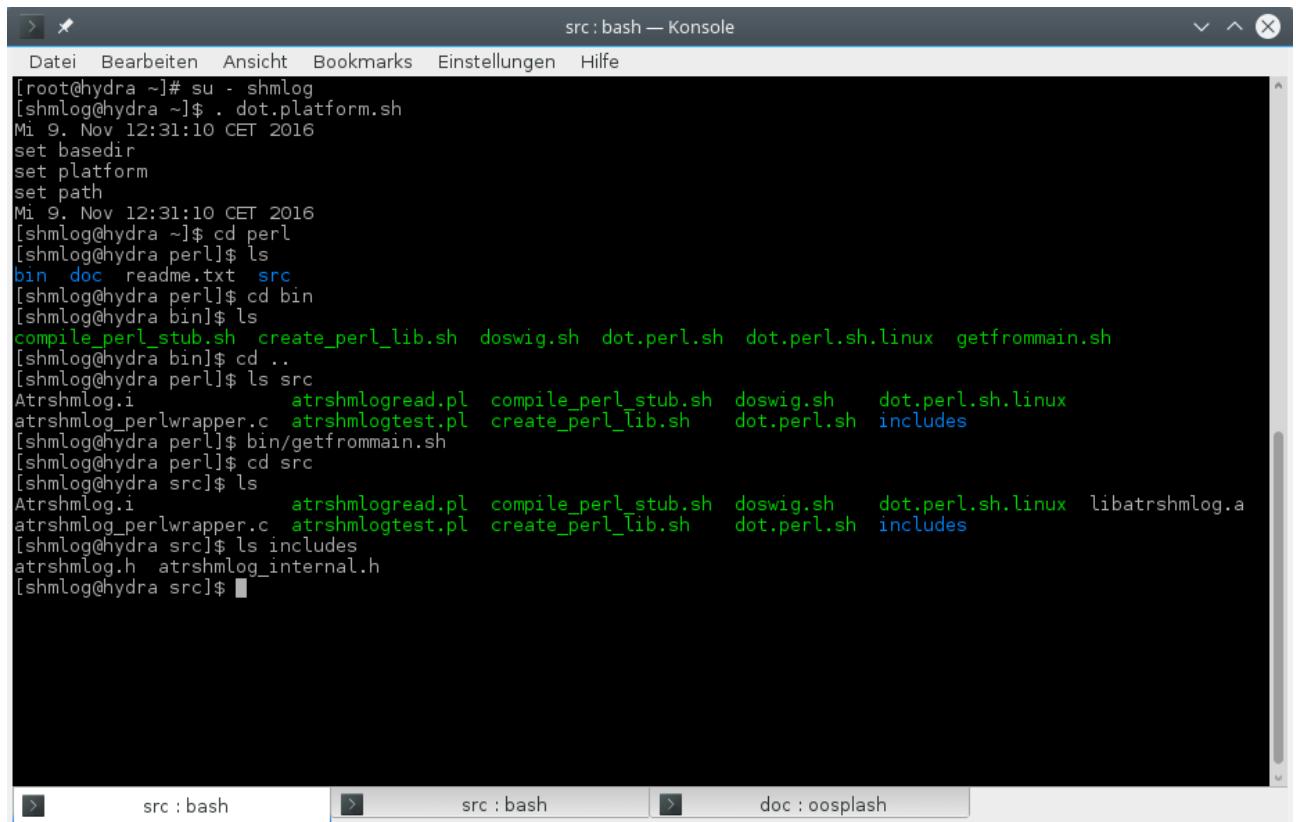
*Illustration 55: Transfer of library and headers*

Not much noise. Well, we will see....

Now its time to check for the src directory's.

## **Change into your src directory**

We change and then see for it



The screenshot shows a terminal window titled "src : bash — Konsole". The window contains a command-line session where the user has navigated into the "src" directory. The session starts with "su - shmlog" to become root, followed by ". dot.platform.sh" to set up the environment. Then, the user changes to the "perl" directory ("cd perl") and lists its contents ("ls"). The output shows several files: "bin", "doc", "readme.txt", "src", "compile\_perl\_stub.sh", "create\_perl\_lib.sh", "doswig.sh", "dot.perl.sh", "dot.perl.sh.linux", "getfrommain.sh", "atrshmlog.i", "atrshmlogread.pl", "compile\_perl\_stub.sh", "doswig.sh", "dot.perl.sh.linux", "atrshmlog\_perlwrapper.c", "atrshmlogtest.pl", "create\_perl\_lib.sh", "dot.perl.sh", "includes", "bin/getfrommain.sh", "atrshmlog.h", "atrshmlog\_internal.h", and "libatrshmlog.a". Finally, the user lists the "src" directory again ("ls src") and exits ("exit").

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mi 9. Nov 12:31:10 CET 2016
set basedir
set platform
set path
Mi 9. Nov 12:31:10 CET 2016
[shmlog@hydra perl]$ cd perl
[shmlog@hydra perl]$ ls
bin doc readme.txt src
[shmlog@hydra perl]$ cd bin
[shmlog@hydra bin]$ ls
compile_perl_stub.sh create_perl_lib.sh doswig.sh dot.perl.sh dot.perl.sh.linux getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra perl]$ ls src
Atrshmlog.i atrshmlogread.pl compile_perl_stub.sh doswig.sh dot.perl.sh.linux
atrshmlog_perlwrapper.c atrshmlogtest.pl create_perl_lib.sh dot.perl.sh includes
[shmlog@hydra perl]$ bin/getfrommain.sh
[shmlog@hydra perl]$ cd src
[shmlog@hydra src]$ ls
Atrshmlog.i atrshmlogread.pl compile_perl_stub.sh doswig.sh dot.perl.sh.linux libatrshmlog.a
atrshmlog_perlwrapper.c atrshmlogtest.pl create_perl_lib.sh dot.perl.sh includes
[shmlog@hydra src]$ ls includes
atrshmlog.h atrshmlog_internal.h
[shmlog@hydra src]$ exit
```

*Illustration 56: Inside the source ready for build*

The src is the real thing. We have again the – adapted - scripts here, so we are free to change them if we need. For reference there are the scripts in bin, so don't change them in bin till you have done the whole thing. Start only in src with changing.

We switch to src and now comes the list of the files in there.

- **compile\_perl\_stub.sh**  
As already said. The main build script..
- **create\_perl\_lib.sh**  
The helper to compile the bridge and link.
- **dot.perl.sh**  
The setting of the environment variables. Next thing to do.
- **doswig.sh**  
The helper to make the SWIG generator run.

- Atrshmlog.i  
The SWIG definition file for this module.
- atrshmlog\_perlwrapper.c  
The perl bridge with helpers code.
- dot.perl.sh.linux  
The setting of the environment variables for Linux.
- libatrshmlog.a  
The copy of the library from the C module (check time stamp and size and whatever you need too ... should be from BASEDIR/src )
- atrshmlogtest.pl  
Start the simple test.
- atrshmlogread.pl  
Helper to start the simple test with read\_fetch.

For the directory's : includes is clear – check for the files and for the time stamp and length of the headers from BASEDIR/src.

OK. Now that we know that tree, check for the others.

When you are ready, you can read on.

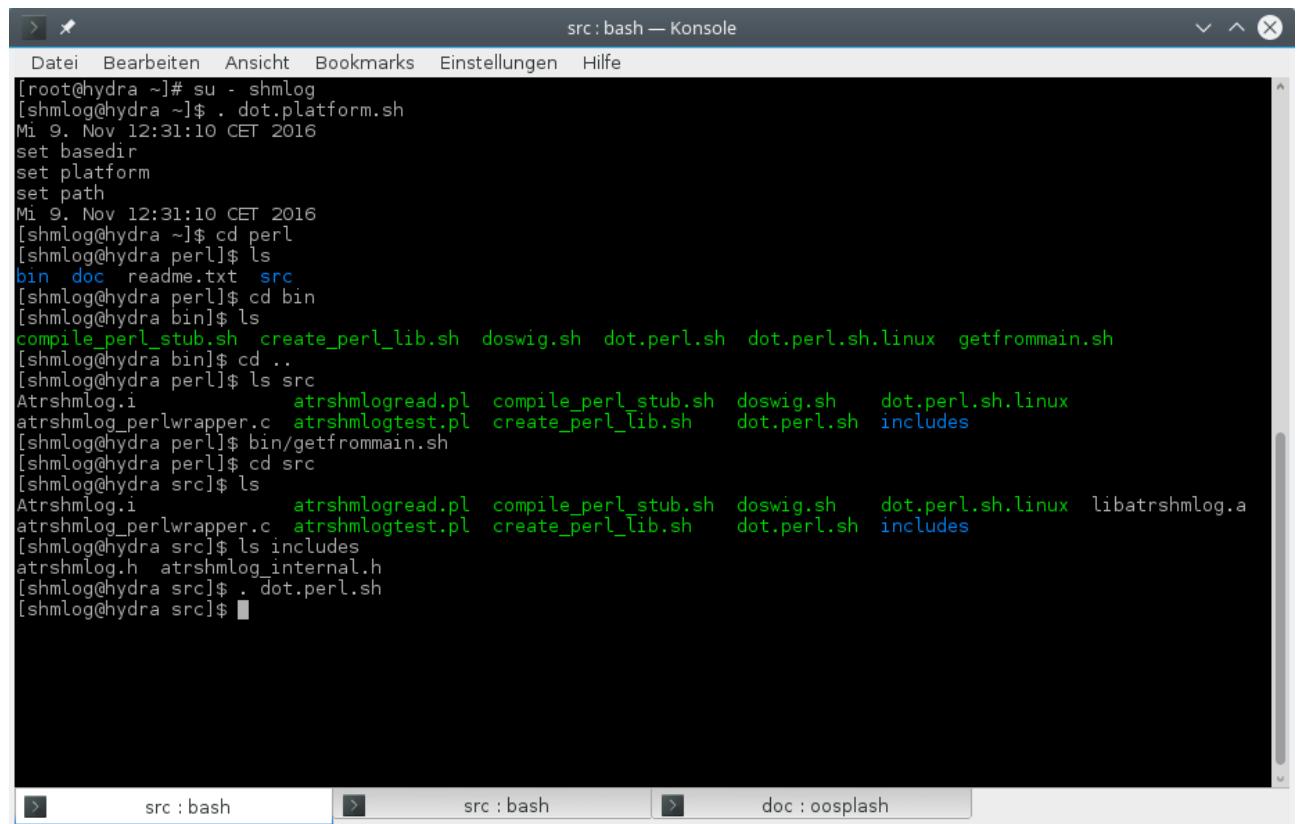
Ready ? So fast – well, if you said so.

Its now time to do the thing.

## **Setting the environment**

Before you do it you need an already up environment for the platform. In doubt switch to BASEDIR and source the dot file you need.

We then simply source dot.perl.sh



The screenshot shows a terminal window titled "src : bash — Konsole". The window contains a command-line session where a user is setting up an environment. The user runs "su - shmlog" to become the shmlog user, then executes ". dot.platform.sh". This script sets several environment variables: "set basedir", "set platform", and "set path". It also changes to the perl directory ("cd perl") and lists files there ("ls"). It then changes to the bin directory ("cd bin") and lists files there ("ls"). The script runs several commands: "compile\_perl\_stub.sh", "create\_perl\_lib.sh", "doswig.sh", "dot.perl.sh", "dot.perl.sh.linux", and "getfrommain.sh". It also creates a directory "src" and lists files in it ("ls src"). Finally, it runs "atrshmlog.i", "atrshmlogread.pl", "compile\_perl\_stub.sh", "doswig.sh", "dot.perl.sh.linux", and "libatrshmlog.a". The session ends with ". dot.perl.sh" and a final "ls" command in the src directory.

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Mi 9. Nov 12:31:10 CET 2016
set basedir
set platform
set path
Mi 9. Nov 12:31:10 CET 2016
[shmlog@hydra ~]$ cd perl
[shmlog@hydra perl]$ ls
bin doc readme.txt src
[shmlog@hydra perl]$ cd bin
[shmlog@hydra bin]$ ls
compile_perl_stub.sh create_perl_lib.sh doswig.sh dot.perl.sh dot.perl.sh.linux getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra bin]$ ls src
Atrshmlog.i atrshmlogread.pl compile_perl_stub.sh doswig.sh dot.perl.sh.linux
atrshmlog_perlwrapper.c atrshmlogtest.pl create_perl_lib.sh dot.perl.sh includes
[shmlog@hydra bin]$ bin/getfrommain.sh
[shmlog@hydra perl]$ cd src
[shmlog@hydra src]$ ls
Atrshmlog.i atrshmlogread.pl compile_perl_stub.sh doswig.sh dot.perl.sh.linux libatrshmlog.a
atrshmlog_perlwrapper.c atrshmlogtest.pl create_perl_lib.sh dot.perl.sh includes
[shmlog@hydra src]$ ls includes
atrshmlog.h atrshmlog_internal.h
[shmlog@hydra src]$ . dot.perl.sh
[shmlog@hydra src]$
```

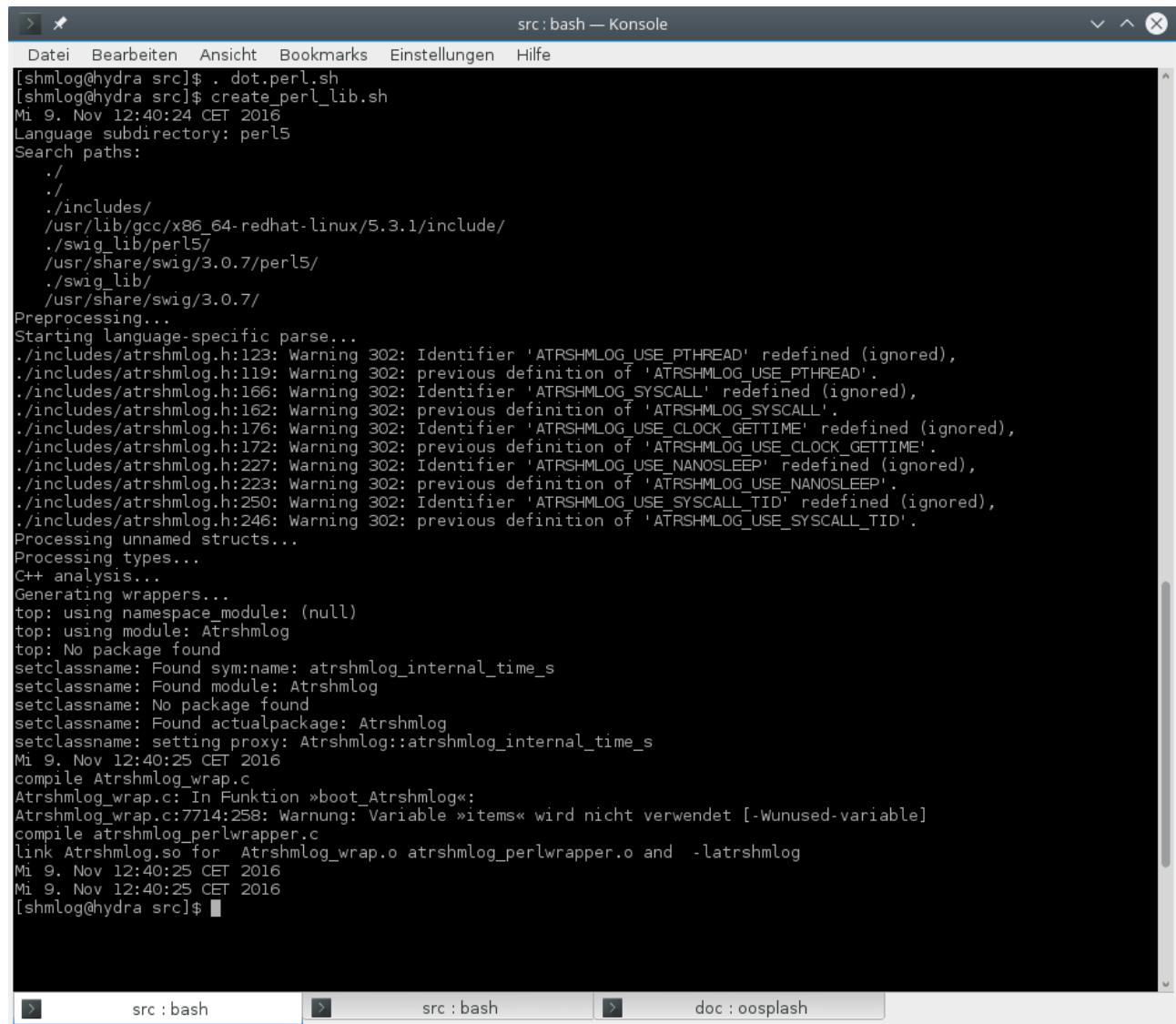
*Illustration 57: Setting the environment for build*

No noise here. If you insist you can check the environment.

Next is to start the build.

## ***Building with create\_perl\_lib.sh***

We start the script



```
[shmlog@hydra src]$ . dot.perl.sh
[shmlog@hydra src]$ create_perl_lib.sh
Mi 9. Nov 12:40:24 CET 2016
Language subdirectory: perl5
Search paths:
./
./includes/
/usr/lib/gcc/x86_64-redhat-linux/5.3.1/include/
./swig_lib/perl5/
/usr/share/swig/3.0.7/perl5/
./swig_lib/
/usr/share/swig/3.0.7/
Preprocessing...
Starting language-specific parse...
./includes/atrshmlog.h:123: Warning 302: Identifier 'ATRSHMLOG_USE_PTHREAD' redefined (ignored),
./includes/atrshmlog.h:119: Warning 302: previous definition of 'ATRSHMLOG_USE_PTHREAD'.
./includes/atrshmlog.h:166: Warning 302: Identifier 'ATRSHMLOG_SYSCALL' redefined (ignored),
./includes/atrshmlog.h:162: Warning 302: previous definition of 'ATRSHMLOG_SYSCALL'.
./includes/atrshmlog.h:176: Warning 302: Identifier 'ATRSHMLOG_USE_CLOCK_GETTIME' redefined (ignored),
./includes/atrshmlog.h:172: Warning 302: previous definition of 'ATRSHMLOG_USE_CLOCK_GETTIME'.
./includes/atrshmlog.h:227: Warning 302: Identifier 'ATRSHMLOG_USE_NANOSLEEP' redefined (ignored),
./includes/atrshmlog.h:223: Warning 302: previous definition of 'ATRSHMLOG_USE_NANOSLEEP'.
./includes/atrshmlog.h:250: Warning 302: Identifier 'ATRSHMLOG_USE_SYSCALL_TID' redefined (ignored),
./includes/atrshmlog.h:246: Warning 302: previous definition of 'ATRSHMLOG_USE_SYSCALL_TID'.
Processing unnamed structs...
Processing types...
C++ analysis...
Generating wrappers...
top: using namespace_module: (null)
top: using module: Atrshmlog
top: No package found
setclassname: Found sym:name: atrshmlog_internal_time_s
setclassname: Found module: Atrshmlog
setclassname: No package found
setclassname: Found actualpackage: Atrshmlog
setclassname: setting proxy: Atrshmlog::atrshmlog_internal_time_s
Mi 9. Nov 12:40:25 CET 2016
compile Atrshmlog_wrap.c
Atrshmlog_wrap.c: In Funktion »boot_Atrshmlog«:
Atrshmlog_wrap.c:7714:258: Warnung: Variable »items« wird nicht verwendet [-Wunused-variable]
compile atrshmlog_perlwrapper.c
link Atrshmlog.so for Atrshmlog_wrap.o atrshmlog_perlwrapper.o and -latrshmlog
Mi 9. Nov 12:40:25 CET 2016
Mi 9. Nov 12:40:25 CET 2016
[shmlog@hydra src]$
```

*Illustration 58: Build the perl library*

And there we are.

First it starts the SWIG generator for a perl module. After the swig is done its compile time.

One compile for the atrshmlog\_wrap.c .

Next compile for the atrshmlog\_perlwrapper.c .

And one link line.

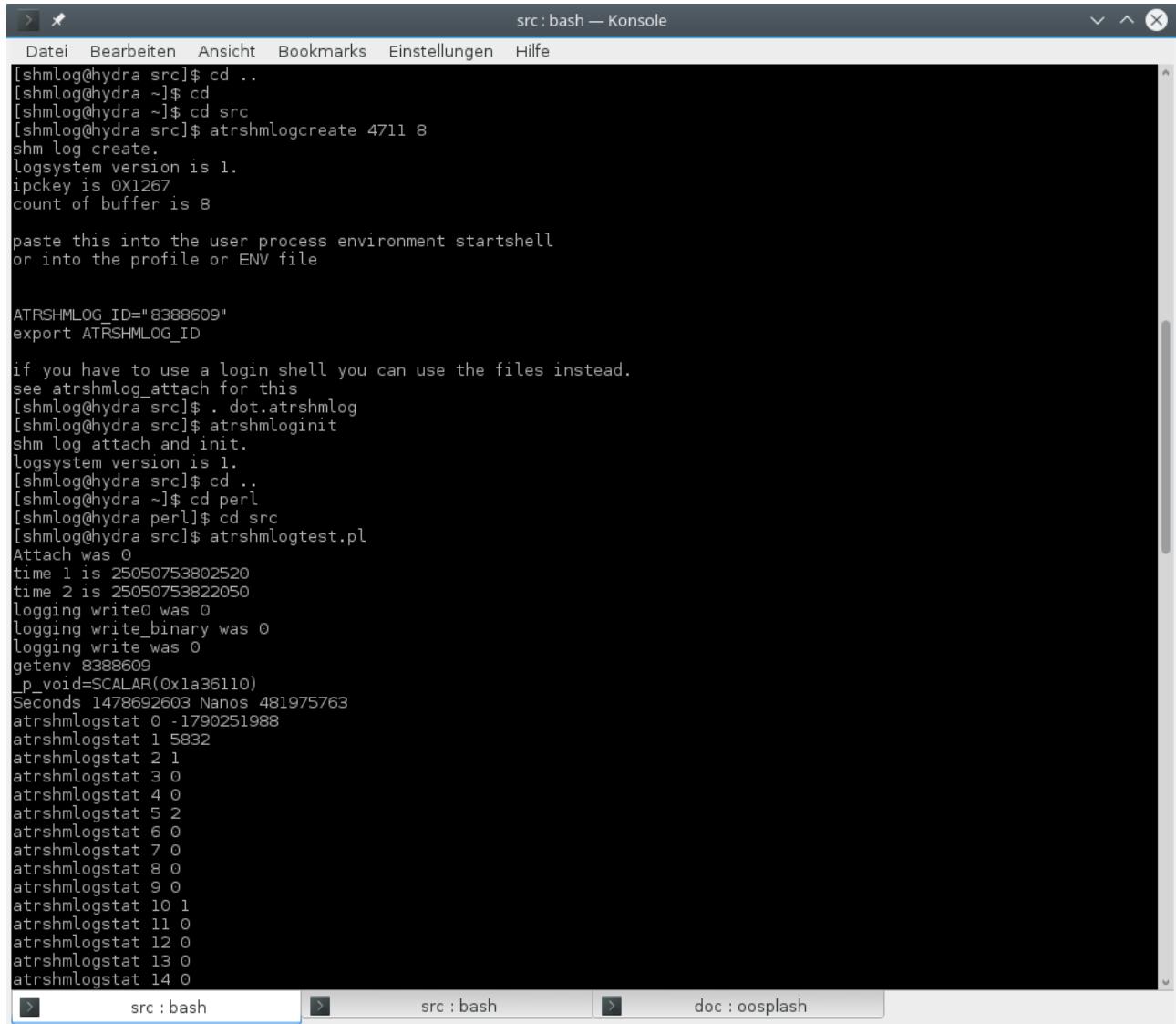
All what's left is the test.

## **Testing the perl bridge**

We have first to create a shared memory buffer with atrshmlogcreate. I will use 4711 and 8 for it. Then the init for the area. Next the test of the bridge. Then the reader and the convert.

At last we can check for the result log.

And because it was already done for the C test program we do it in short form now.



```
[shmlog@hydra src]$ cd ..
[shmlog@hydra ~]$ cd
[shmlog@hydra ~]$ cd src
[shmlog@hydra src]$ atrshmlogcreate 4711 8
shm log create.
logsystem version is 1.
ipckey is 0x1267
count of buffer is 8

paste this into the user process environment startshell
or into the profile or ENV file

ATRSHMLOG_ID="8388609"
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
[shmlog@hydra src]$ . dot.atrshmlog
[shmlog@hydra src]$ atrshmloginit
shm log attach and init.
logsystem version is 1.
[shmlog@hydra src]$ cd ..
[shmlog@hydra ~]$ cd perl
[shmlog@hydra perl]$ cd src
[shmlog@hydra src]$ atrshmlogtest.pl
Attach was 0
time 1 is 25050753802520
time 2 is 25050753822050
logging write0 was 0
logging write_binary was 0
logging write was 0
getenv 8388609
_p_void=SCALAR(0xa36110)
Seconds 1478692603 Nanos 481975763
atrshmlogstat 0 -1790251988
atrshmlogstat 1 5832
atrshmlogstat 2 1
atrshmlogstat 3 0
atrshmlogstat 4 0
atrshmlogstat 5 2
atrshmlogstat 6 0
atrshmlogstat 7 0
atrshmlogstat 8 0
atrshmlogstat 9 0
atrshmlogstat 10 1
atrshmlogstat 11 0
atrshmlogstat 12 0
atrshmlogstat 13 0
atrshmlogstat 14 0
```

*Illustration 59: Test of the perl library*

HM. Worked as expected.

And here the result for the log after the fetch from the reader.

```

src : bash — Konsole
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
atrshmlogstat 98 0
atrshmlogstat 99 0
[shmlog@hydra src]$ [shmlog@hydra src]$ cd
[shmlog@hydra ~]$ cd src
[shmlog@hydra src]$ atrshmlogstopreader
shm log attach and set reader flag and pid.
logsystem version is 1.
pid before 0
flag before 0
[shmlog@hydra src]$ atrshmlogreaderd dl
shm log attach and loop write file.
logsystem version is 1.
directory is dl
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'dl'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
logging done.
writer data : time 1181820 count 1 per use 1181820
[shmlog@hydra src]$ atrshmlogconv dl
shm log converter from file 'd1/0/atrshmlog_p11745_t11745_s0_f0.bin' to file 'd1/0/atrshmlog_p11745_t11745_s0_f0.txt'.
id 1 acquiretime 1480 pid 11745 tid 11745 slavetime 1220 readertime 9200 payloadszie 117 sh
mbuffer 0 filenumber 0 sequence 0
[shmlog@hydra src]$ cat d1/0/atrshmlog_p11745_t11745_s0_f0.txt
0000011745 0000000000011745 000 0000000000000000 000025050753802520 000025050753822050 0000000000000019530 14786926
03481918065 1478692603481925384 00000000000000007319 00000000001 I 0000000042
0000011745 0000000000011745 000 0000000000000000 000025050753802520 000025050753822050 0000000000000019530 14786926
03481918065 1478692603481925384 00000000000000007319 00000000001 I 0000000043 hello, world
0000011745 0000000000011745 000 0000000000000000 000025050753802520 000025050753822050 0000000000000019530 14786926
03481918065 1478692603481925384 00000000000000007319 00000000001 I 0000000044 hello, anton world
[shmlog@hydra src]$ █

```

The terminal window shows the execution of the `atrshmlogstat` command followed by the `atrshmlogconv` command with the argument `dl`. The output of `atrshmlogconv` shows the conversion of a binary log file (`d1/0/atrshmlog_p11745_t11745_s0_f0.bin`) into a text file (`d1/0/atrshmlog_p11745_t11745_s0_f0.txt`). The text file contains log entries with fields such as id, acquiretime, pid, tid, slavetime, readertime, payloadszie, and sequence. The final line shows a log entry with the message "hello, anton world".

*Illustration 60: Test log output*

OK. So we have the log in here.

## Details

Now we make it for the bridge in C.

Take your favorite text editor and open the `Atrshmlog.i`.

The file is not so complicated. Its only a bit bigger than a good SWIG example because of the rough hundred functions we have in here.

It starts with the usual comment stuff.

Then the definition of the module. I use a Camel case here like in most modules in perl.

Next is the code block for the wrapper file with the usual include of the C module.

What follows are the prototypes of the helpers that we will need.

There is only small need for helpers here – compared to the python its a really thin layer.

We need the otherwise inlined gettimeofday.

And for the SWIG way to handle output the helpers of `get_inittime`, `get_readtime` and `get_statistics`.

The SWIG way is done by conversion of output pointer parameters into temporary variables – and

then to give back them on the perl stack which makes them a simple return array.

So we get an array with two values from the get\_inittime and the get\_realtimetime.

For the get\_statistics its a array of – well – hundred values for now. I simply say that's OK for me, after all I have hundred counters in place. And only 86 are in use now.

Last are the read and read\_fetch helpers. They need a buffer and I have made it a full length one.

So with the small hacks for peek and poke the included code finish.

Next is the typemap include. I had a strange result when I had a comment after the thing – so I removed the comment and anything is now back to normal.

Its for the OUTPUT parameters, see later.

Then we have the typemap for the c string buffer handling in read and read\_fetch.

I had some trouble with two things in the first place, so I added two defines to make the compiler work again.

Perhaps this is not needed for your SWIG implementation.

The n the ignore block follows. I simply cannot use the raw prototypes, so I use the include of the interface and the raw from the internal to make the SWIG see my C module functions.

This has a downside : I had to make for some variables a SWIG directive to NOT get some silly setter stuff.

And I catched some stuff from stdint.h here in too.

So that for the ignore block.

Next the translation of the names follow. Most is to cut of the atrshmlog\_ prefix. For the helpers its the additional atratrshmlog\_ prefix.

After that is set the include of the interface header is done.

Last are the prototypes of the helpers and the modifications to make the OUTPUT and cstring buffer things to work.

Now we can switch to the wrapper helpers code.

Open the atrshmlog\_perlwrapper.c with your favorite text editor.

We start with the usual comment stuff.

Then the includes. We need a strlen in the write, so we have also the string.h header here.

First is gettimeofday. Its more or less the Macro version here.

Next is write binary. We get at least a address and the length in here.

Then the heart of the client log, the write. We use a string here and take its length. No binary in

here. If you need binary use the write\_binary instead.

Next are the two getter for times. We need to make the output thing work, so no struct as return value in the perl layer. Instead we get with the OUTPUT typemap thing here two values on the return stack – which makes a array with two values for me. Perfect.

Then the get statistics. Again its only about a different interface and again the OUTPUT is used. This time you get 100 values in the return array.

For read and read\_fetch we use a thread local buffer – its a mallocoed one and a big one.

So if you really plan to do the read or read\_fetch thing you will need memory, and if you use multiple threads its for every thread an additional Mib.

We already made it for peek and poke in the definition file. So no implementation here.

And this is it. We are through alrea.... Ahem. OK.

The SWIG generated too.

I started after the XS didn't make it.

So first thing first – get an example and see how its done.

Swig's home page is OK for it. And it was a very simple one.

I checked also some things on google and found this should be easy. So I made the first approach.

And - BANG. Got compile errors for things I did not use in the first place – that's frustrating, crashing because of a thing you didn't need.

First was the conversion of a wchar\_t value – the generated function was simply not there.

So because I even don't need it I made a define hack in the compile script.

Next was a missing off64\_t. OK, next define.

After an hour I was up and running.

Next was to get info about the get\_statistics thing. I got a scalar reference – nothing you can use in perl for the thing.

I checked google and found a nice article for perl98 about a return of arrays in C for perl. The thing was easy to make, some simple helper to alloc mem, then do the call of get\_statistics with the returned scalar reference, then use some getter function to do the fetches into an perl array – and then delete the no longer allocoed in between array.

Sounds great – for a C developer.

For a perl user after 20 years, well, horrible.

So I checked again when I made the get\_inittime. And this time I found the real thing. A function that uses an OUTPUT in the prototype delivers a value back. Having more than one OUTPUT simply delivers multiple values. And for the types I had it was the simple number stuff – and so I

got an array with two values back.

That was for a C developer a bit frustrating, but a helper did the job.

For the perl user it was - well, as expected. No big hooray or so. Simple use it.

Back to get\_statistics.

Make a helper, use the array in between and see the thing was done in the wrap.c again. And it was OK.

Nice. Now the last three hours – the read and read\_fetch.

Again the helper was needed because of the stuff that is delivered via pointers to the caller. I needed the OUTPUT again.

For the major part, the buffer, I had again to check. And I found a thing that worked for binary and did the allocation and the free in perl itself. OK, this was what I had in mind. For python the code was done in the core functions, so I simply made copy's and changed parts of them. After the three hours I had a working read\_fetch demo.

Last hour. Cleanup the no longer needed. Make some helpful constants and rename the things.

Again the help of SWIG.org did the job.

And so we have now a SWIG generated wrapper.

Check the doswig.sh for the conversion and for a small hack for a erroneous construct that I made here.

Now open the Attrshmlog\_wrap.c in your favorite text editor.

You find at the beginning the code for my two problematic compiler no-go.

Then you can skip the swig machinery code.

It about line 2100 when we are back in the module.

The SWIG first handles some variables and return struct things. I don't need them, but SWIG thinks it could not hurt.

For the variable's I tolerate the getter's. But the setters - I don't need them. So you are having now some immutable directives in the interface header to block that.

After the setters and getter's we enter the REAL stuff in 2300.

Attach.

Its a simple one, no parameters, only a return value.

And yes, its actual an int in Linux land here. So we get a simple wrapper.

Ignore the get\_statistics here, its dead.

Next is the write0 . Some more parameters, but that's it.

Sleep nanos is simple, only a parameter, and again its an int for SWIG and we are in.

As you can see it is no big deal to read it. But to write would be a lot more work. So I am happy the generator works....

After the environment handling stuff with strings we enter the big number of simple setter and getter like functions.

Most look like the first two of them.

Some minor things to mention.

The area things are done in the jni and python by using numbers. Here they are scalar references.

About line 4400 we hit the otherwise inlined functions from the interface .

What follows are the prototypes from the definition file for the create, the delete, cleanup\_locks and init\_shm\_log. After this we have the helpers in place.

The gettime is easy. The write\_binary has some more to do. But again only parameter stuff.

With the get\_inittime we enter the magic of OUTPUT. It simply works, so I don't question it.

Then the real get\_statistics is in.

Its a big – but otherwise boring – version of the get time things.

And that's it for the wrap.c

Now the last thing we need – and the most important for the perl user – is the generated pm file.

Fire up your editor and load it in.

To make things easy you can skip the first lines. From line 50 on we have our new functions for the module.

The next we have is the not needed stuff for the return structs.

With line 200 we are in again. SWIG makes heavy use of structs and defines and constants and enum's in the generation, so we have them all in.

To be honest its a bit too much in my experience, but if it does not hurt I take it.

So you can now import it – have the shared lib and the compiler libs in place – and use it.

## The one fits all SWIG approach

After I had done the initial jni stuff it was clear for me to make it to other languages too.

So on my first list I had python, perl and for those others SWIG in case they use that.

For the perl I had to start at with my python experience.

I tried the standard perl way, the XS, but failed. So I switched to SWIG – which I had on my list for later anyway. And it worked out.

So my SWIG was already done. But I don't know if I will change to XS back and so this is the vanilla SWIG chapter.

I chose as target the tcl – another now over 20 years well known language to support the embedding and to be embedded in other systems thing.

So we will ignore the perl swig solution for now and simply start from ground zero.

If you need another language you should do some experiments with the SWIG to find out if it supports your language as it did for perl and tcl.

## How it works

The SWIG uses its definition file to build a wrapper C code file in tcl. No other files are in. So you have in place the swig wrapper, the additional C code file with the helpers and the C module itself.

The module is a direct used implementation. Under cover you use the C module. The module is capable of doing its job simply by using the internal infrastructure as in a C environment. No need for changes inside.

This means you get in theory the same speed as for the C module itself. The only thing that slows down is the tcl parameter handling in the bridge code itself.

The rest is more or less simple. Use of the function and no object stuff. Simple raw data types, no classes, no exceptions and no use of the tcl side. Barely some string creations.

If you are interested the atrshmlog\_swigwrapper.c is a good start.

The module gives also the a small number of constants to the user.

Also the enum's are there, but here you have to use the C notation with the in C usual prefix.

A simple load gets it in.

## How to use it

The user needs the following parts:

- Shared library or dll Atrshmlog

This is the bridge code from the SWIG code atrshmlog\_wrap.c , the atrshmlog\_swigwrapper.c and the linked library from the C module, libatrshmlog.a. You need also the compiler support lib's and in case of the mingw port the additional dll 's of the mingw system.

- The support programs

You need the matching C support programs for the basic stuff you don't want to reinvent. Reader, converter, even create and delete. So if you need a pure (?) own language only solution you can replace them with your own code counterparts. But for now I deliver nothing so you should start with the C programs.

So you first have to build it – again I am not delivering binary code, only source, so if you are interested in binary you have to contact me and I will see what I can do. But for now you have to start with the SWIG's bridge file atrshmlog\_swigwrapper.c and Attrshmlog.i.

## **How to build it in the first place**

We start with the C module.

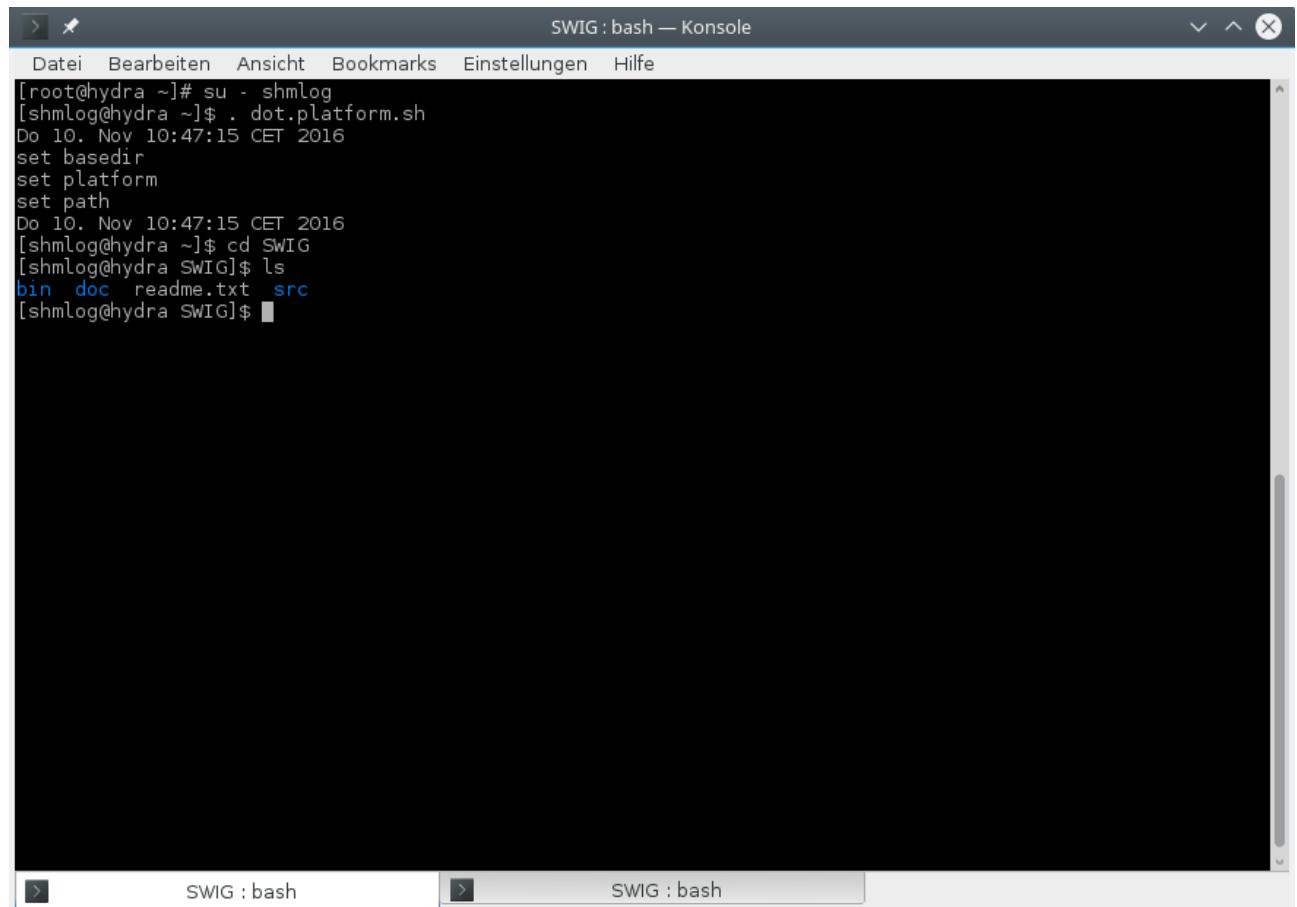
After this is in place we have to check our next options.

We have in place an implementation that should work out of the box for Linux and tcl 8.6.

## **The SWIG directory**

We start with the basics.

The SWIG stuff is located parallel to src – so we start at BASEDIR/SWIG



The screenshot shows a terminal window titled "SWIG : bash — Konsole". The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Bookmarks", "Einstellungen", and "Hilfe". The terminal output is as follows:

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Do 10. Nov 10:47:15 CET 2016
set basedir
set platform
set path
Do 10. Nov 10:47:15 CET 2016
[shmlog@hydra ~]$ cd SWIG
[shmlog@hydra SWIG]$ ls
bin doc readme.txt src
[shmlog@hydra SWIG]$
```

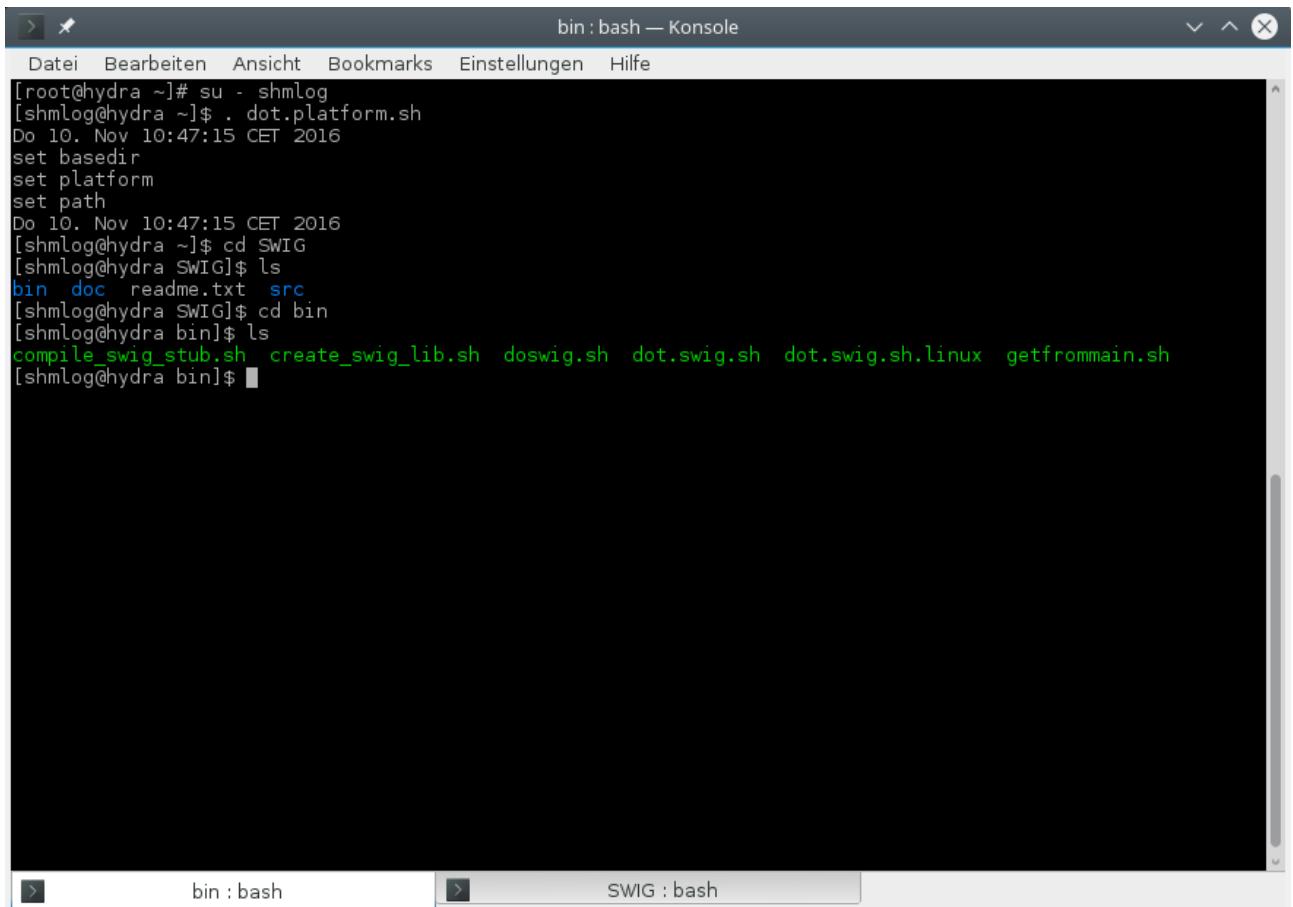
*Illustration 61: The SWIG base directory*

OK. We have a readme – check it – and a bin, a doc and a src.

For the bin its clear, there are some scripts.

## The bin directory

That's my bin so far.



A screenshot of a terminal window titled "bin : bash — Konsole". The window shows a command-line session where the user is navigating through a directory structure. The session starts with "su - shmlog" to become root, then runs ". dot.platform.sh" which sets up environment variables. It then changes directory to "SWIG", lists its contents, and changes to the "bin" directory. Inside "bin", it lists several scripts: "compile\_swig\_stub.sh", "create\_swig\_lib.sh", "doswig.sh", "dot.swig.sh", "dot.swig.sh.linux", and "getfrommain.sh". The terminal has a dark background with light-colored text and a light gray scroll bar on the right.

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Do 10. Nov 10:47:15 CET 2016
set basedir
set platform
set path
Do 10. Nov 10:47:15 CET 2016
[shmlog@hydra ~]$ cd SWIG
[shmlog@hydra SWIG]$ ls
bin doc readme.txt src
[shmlog@hydra SWIG]$ cd bin
[shmlog@hydra bin]$ ls
compile_swig_stub.sh create_swig_lib.sh doswig.sh dot.swig.sh dot.swig.sh.linux getfrommain.sh
[shmlog@hydra bin]$
```

Illustration 62: The bin directory with the scripts

We have here one script that is interesting now. The rest becomes clear when it comes to a real build cycle. So for now a short info about the scripts

- `compile_swig_stub.sh`  
The main build script – its the same for the SWIG layer as the `makeall.sh` for the C module.
- `create_swig_lib.sh`  
The helper to compile the bridge C code and link to the library to get – ahem – the library ? I think here we have to distinguish the library ( C module ) and the SWIG bridge library for the target languages stuff. We call the later from now on the SWIG library.
- `getfrommain.sh`  
The script to transfer the headers and the library to SWIG directory's. This is the real thing

here, see below.

- doswig.sh

This is the helper to do the swig generation for the SWIG wrapper library..

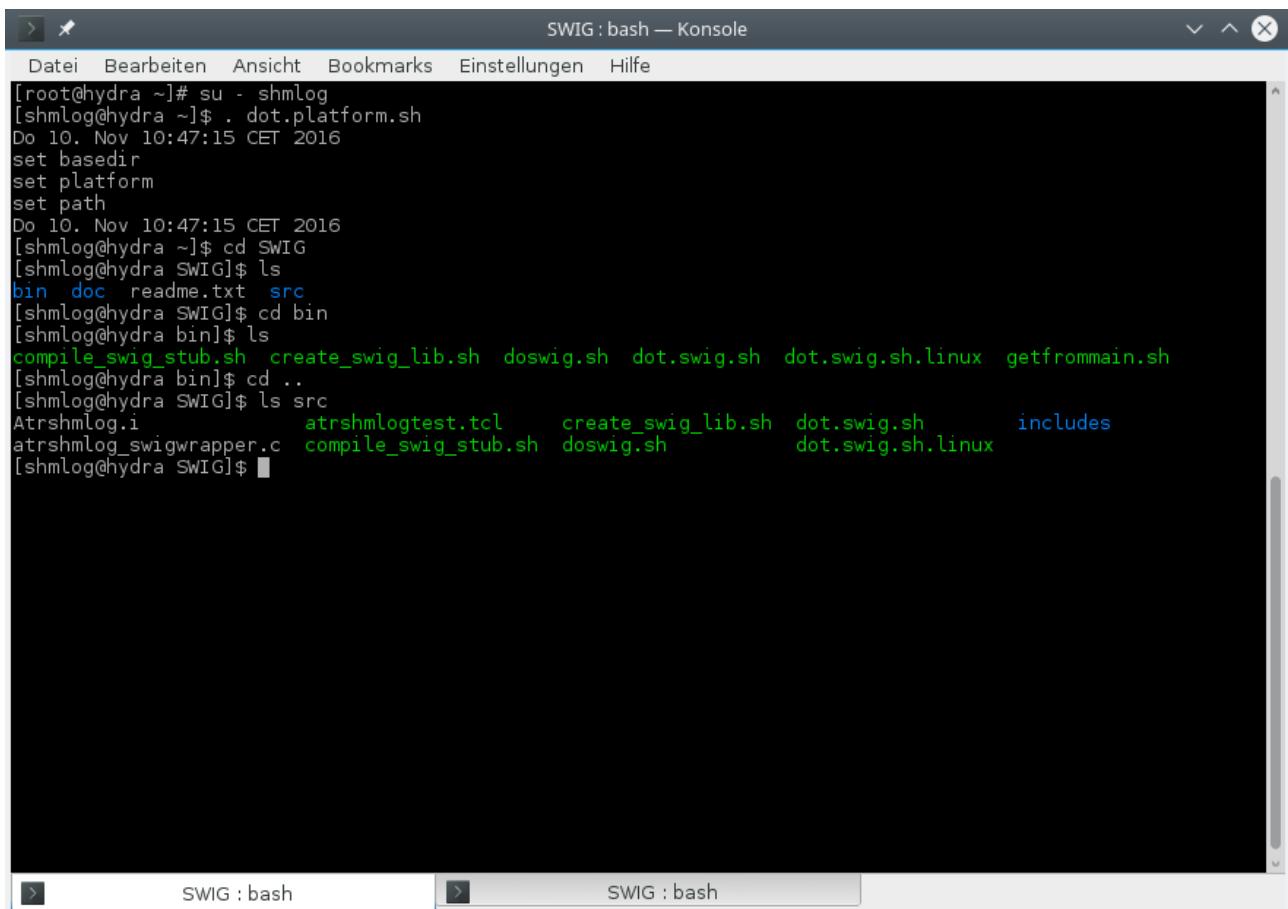
- dot.swig.sh

The environment setter.

- dot.swig.sh.linux

The Linux version – for now the same.

OK. There were one real thing, but its one for now. The getfrommain.sh. It transfers the headers and the library to the mystery SWIG directory's.



The screenshot shows a terminal window titled "SWIG : bash — Konsole". The window has a menu bar with German labels: Datei, Bearbeiten, Ansicht, Bookmarks, Einstellungen, Hilfe. The terminal output is as follows:

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Do 10. Nov 10:47:15 CET 2016
set basedir
set platform
set path
Do 10. Nov 10:47:15 CET 2016
[shmlog@hydra ~]$ cd SWIG
[shmlog@hydra SWIG]$ ls
bin doc readme.txt src
[shmlog@hydra SWIG]$ cd bin
[shmlog@hydra bin]$ ls
compile_swig_stub.sh create_swig_lib.sh doswig.sh dot.swig.sh dot.swig.sh.linux getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra SWIG]$ ls src
atrshmlog.i atrshmlogtest.tcl create_swig_lib.sh dot.swig.sh includes
atrshmlog_swigwrapper.c compile_swig_stub.sh doswig.sh dot.swig.sh.linux
[shmlog@hydra SWIG]$
```

*Illustration 63: The source directory for SWIG*

So that's now the SWIG things, and its the src.

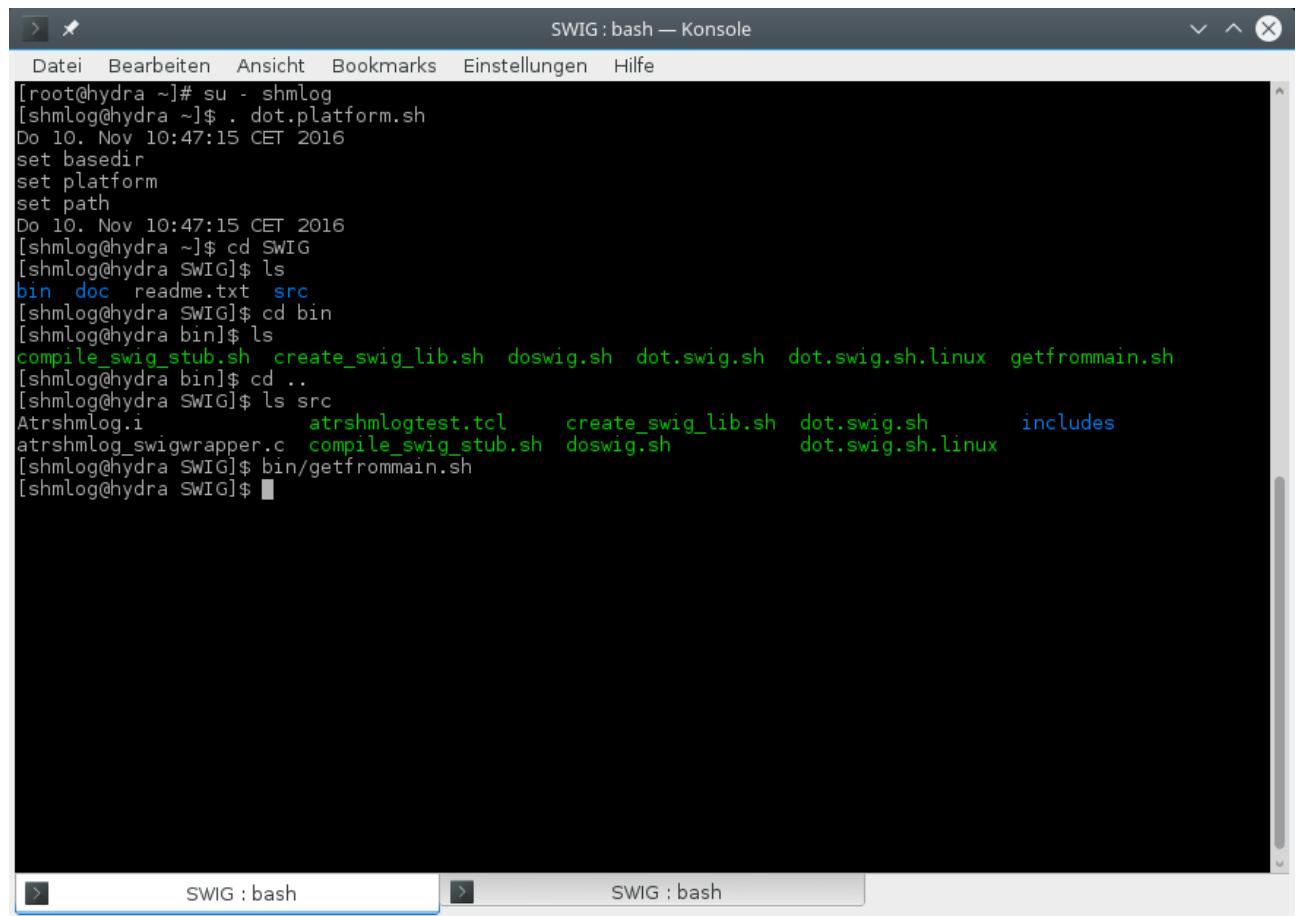
OK. So we transfer with getfrommain.sh from the src what's there to the whole bunch.

Meaning: you should have the real headers in src in place. And the real library. Don't mix platforms or versions – then you have to ignore that script.

If everything is right we can execute the script now. Its mandatory that you are in the SWIG directory for its execution. So we call it relative with bin/getfrommain.sh.

## **Copy headers and lib from the C module**

And so we do it



```
SWIG : bash — Konsole
Datei Bearbeiten Ansicht Bookmarks Einstellungen Hilfe
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Do 10. Nov 10:47:15 CET 2016
set basedir
set platform
set path
Do 10. Nov 10:47:15 CET 2016
[shmlog@hydra ~]$ cd SWIG
[shmlog@hydra SWIG]$ ls
bin doc readme.txt src
[shmlog@hydra SWIG]$ cd bin
[shmlog@hydra bin]$ ls
compile_swig_stub.sh create_swig_lib.sh doswig.sh dot.swig.sh dot.swig.sh.linux getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra SWIG]$ ls src
Atrshmlog.i atrshmlogtest.tcl create_swig_lib.sh dot.swig.sh includes
atrshmlog_swigwrapper.c compile_swig_stub.sh doswig.sh dot.swig.sh.linux
[shmlog@hydra SWIG]$ bin/getfrommain.sh
[shmlog@hydra SWIG]$
```

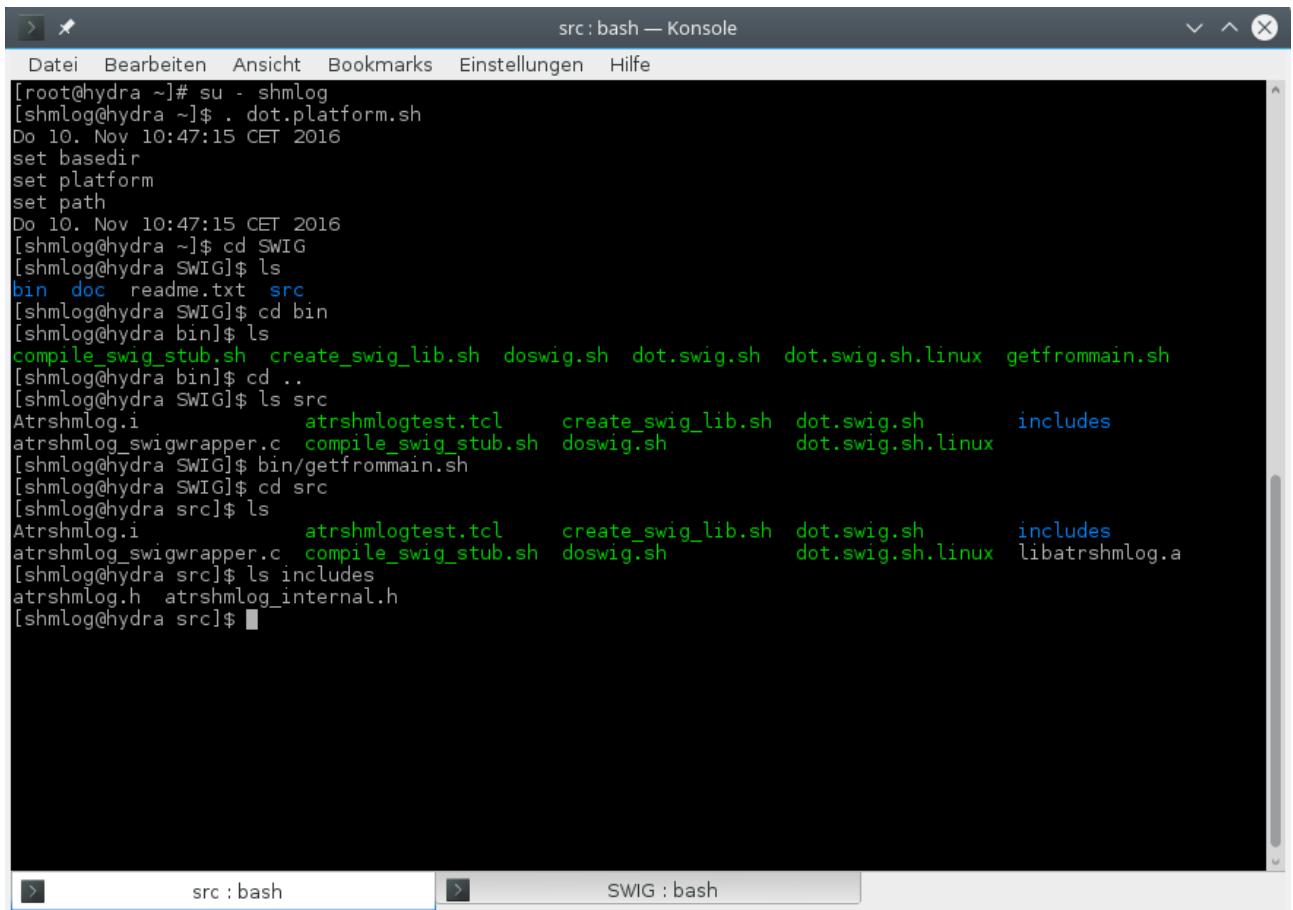
*Illustration 64: Transfer of the library and the headers*

Not much noise. Well, we will see....

Now its time to check for the src directory's.

## **Change into your src directory**

We change and then see for it



The screenshot shows a terminal window titled "src : bash — Konsole". The terminal is running as root on a system named "hydra". The user has run the command ". dot.platform.sh" which sets up environment variables. Then, they navigate to the "SWIG" directory and enter the "src" subdirectory. Inside "src", there are several files: "atrshmlog.i", "atrshmlogtest.tcl", "create\_swig\_lib.sh", "dot.swig.sh", "dot.swig.sh.linux", "includes", "atrshmlog\_swigwrapper.c", and "compile\_swig\_stub.sh". The user also sees "dot.swig.sh" and "dot.swig.sh.linux" again, likely due to symlinks or a build step. Finally, they run "ls includes" and see "atrshmlog.h" and "atrshmlog\_internal.h". The terminal window has a standard Linux-style title bar and a scroll bar.

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Do 10. Nov 10:47:15 CET 2016
set basedir
set platform
set path
Do 10. Nov 10:47:15 CET 2016
[shmlog@hydra ~]$ cd SWIG
[shmlog@hydra SWIG]$ ls
bin doc readme.txt src
[shmlog@hydra SWIG]$ cd bin
[shmlog@hydra bin]$ ls
compile_swig_stub.sh create_swig_lib.sh doswig.sh dot.swig.sh dot.swig.sh.linux getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra SWIG]$ ls src
Atrshmlog.i atrshmlogtest.tcl create_swig_lib.sh dot.swig.sh includes
atrshmlog_swigwrapper.c compile_swig_stub.sh doswig.sh dot.swig.sh.linux
[shmlog@hydra SWIG]$ bin/getfrommain.sh
[shmlog@hydra SWIG]$ cd src
[shmlog@hydra src]$ ls
Atrshmlog.i atrshmlogtest.tcl create_swig_lib.sh dot.swig.sh includes
atrshmlog_swigwrapper.c compile_swig_stub.sh doswig.sh dot.swig.sh.linux libatrshmlog.a
[shmlog@hydra src]$ ls includes
atrshmlog.h atrshmlog_internal.h
[shmlog@hydra src]$
```

*Illustration 65: Inside the source ready for build*

The src is the real thing. We have again the – adapted - scripts here, so we are free to change them if we need. For reference there are the scripts in bin, so don't change them in bin till you have done the whole thing. Start only in src with changing.

We switch to src and now comes the list of the files in there.

- **compile\_swig\_stub.sh**  
As already said. The main build script..
- **create\_swig\_lib.sh**  
The helper to compile the bridge and link.
- **dot.swig.sh**  
The setting of the environment variables. Next thing to do.

- doswig.sh  
The helper to make the SWIG generator run.
- Attrshmlog.i  
The SWIG definition file for this module.
- atrshmlog\_swigwrapper.c  
The SWIG bridge with helpers code.
- dot.swig.sh.linux  
The setting of the environment variables for Linux.
- libatrshmlog.a  
The copy of the library from the C module (check time stamp and size and whatever you need too ... should be from BASEDIR/src )
- atrshmlogtest.tcl  
Start the simple test. For tcl

For the directory's : includes is clear – check for the files and for the time stamp and length of the headers from BASEDIR/src.

OK. Now that we know that tree, check for the others.

When you are ready, you can read on.

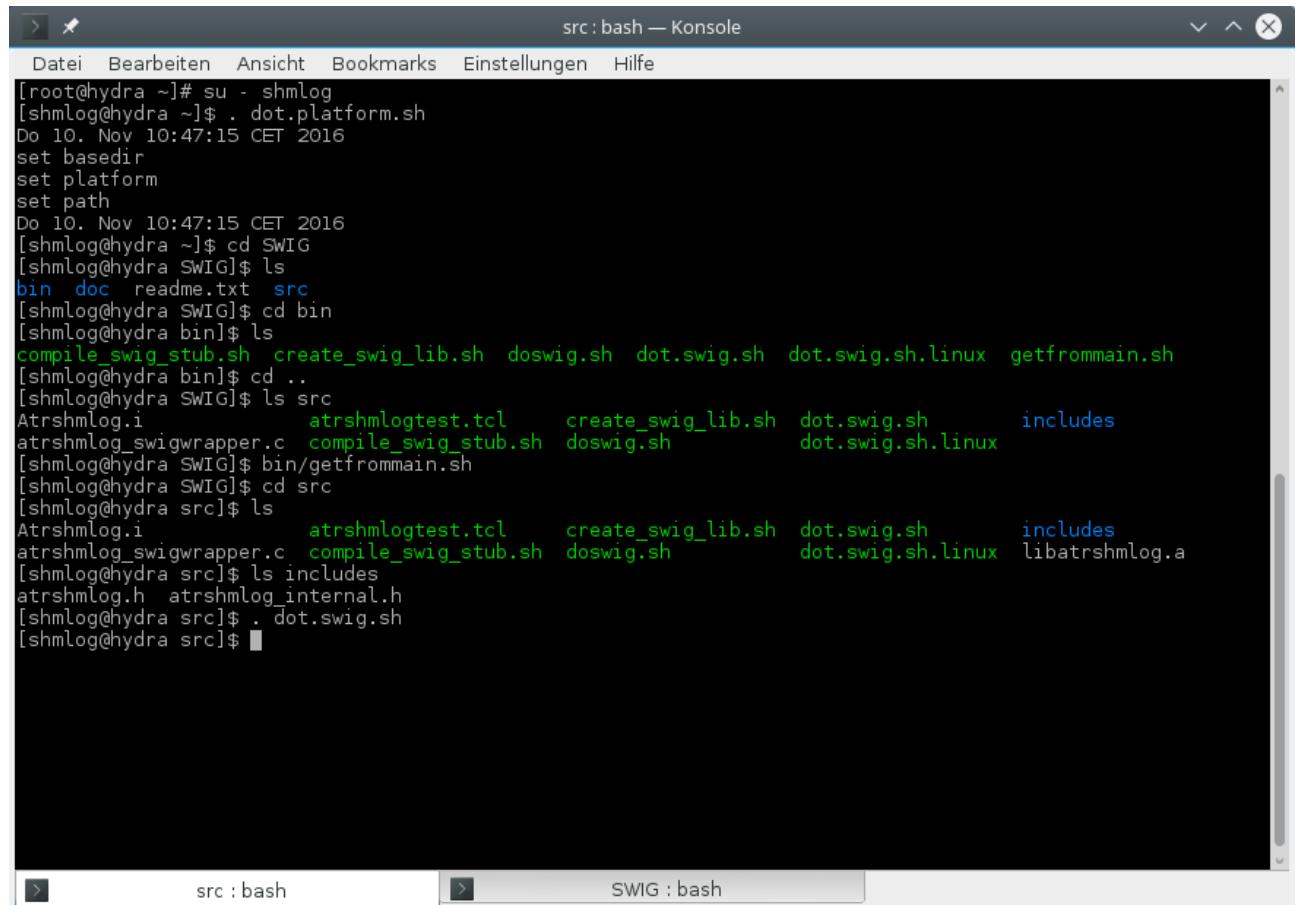
Ready ? So fast – well, if you said so.

Its now time to do the thing.

## **Setting the environment**

Before you do it you need an up environment for the platform. So in doubt first switch to BASEDIR and source the dot file you need.

We then simply source dot.swig.sh



The screenshot shows a terminal window titled "src : bash — Konsole". The terminal is running a series of commands to set up a build environment. The commands include switching to root, sourcing a platform configuration script, setting the base directory, changing to the SWIG directory, listing files in bin and src, and finally running a script to compile a SWIG stub. The terminal window has a dark background with white text and a light gray scroll bar on the right.

```
[root@hydra ~]# su - shmlog
[shmlog@hydra ~]$ . dot.platform.sh
Do 10. Nov 10:47:15 CET 2016
set basedir
set platform
set path
Do 10. Nov 10:47:15 CET 2016
[shmlog@hydra ~]$ cd SWIG
[shmlog@hydra SWIG]$ ls
bin doc readme.txt src
[shmlog@hydra SWIG]$ cd bin
[shmlog@hydra bin]$ ls
compile_swig_stub.sh create_swig_lib.sh doswig.sh dot.swig.sh dot.swig.sh.linux getfrommain.sh
[shmlog@hydra bin]$ cd ..
[shmlog@hydra SWIG]$ ls src
Atrshmlog.i atrshmlogtest.tcl create_swig_lib.sh dot.swig.sh includes
atrshmlog_swigwrapper.c compile_swig_stub.sh doswig.sh dot.swig.sh.linux
[shmlog@hydra SWIG]$ bin/getfrommain.sh
[shmlog@hydra SWIG]$ cd src
[shmlog@hydra src]$ ls
Atrshmlog.i atrshmlogtest.tcl create_swig_lib.sh dot.swig.sh includes
atrshmlog_swigwrapper.c compile_swig_stub.sh doswig.sh dot.swig.sh.linux libatrshmlog.a
[shmlog@hydra src]$ ls includes
atrshmlog.h atrshmlog_internal.h
[shmlog@hydra src]$ . dot.swig.sh
[shmlog@hydra src]$
```

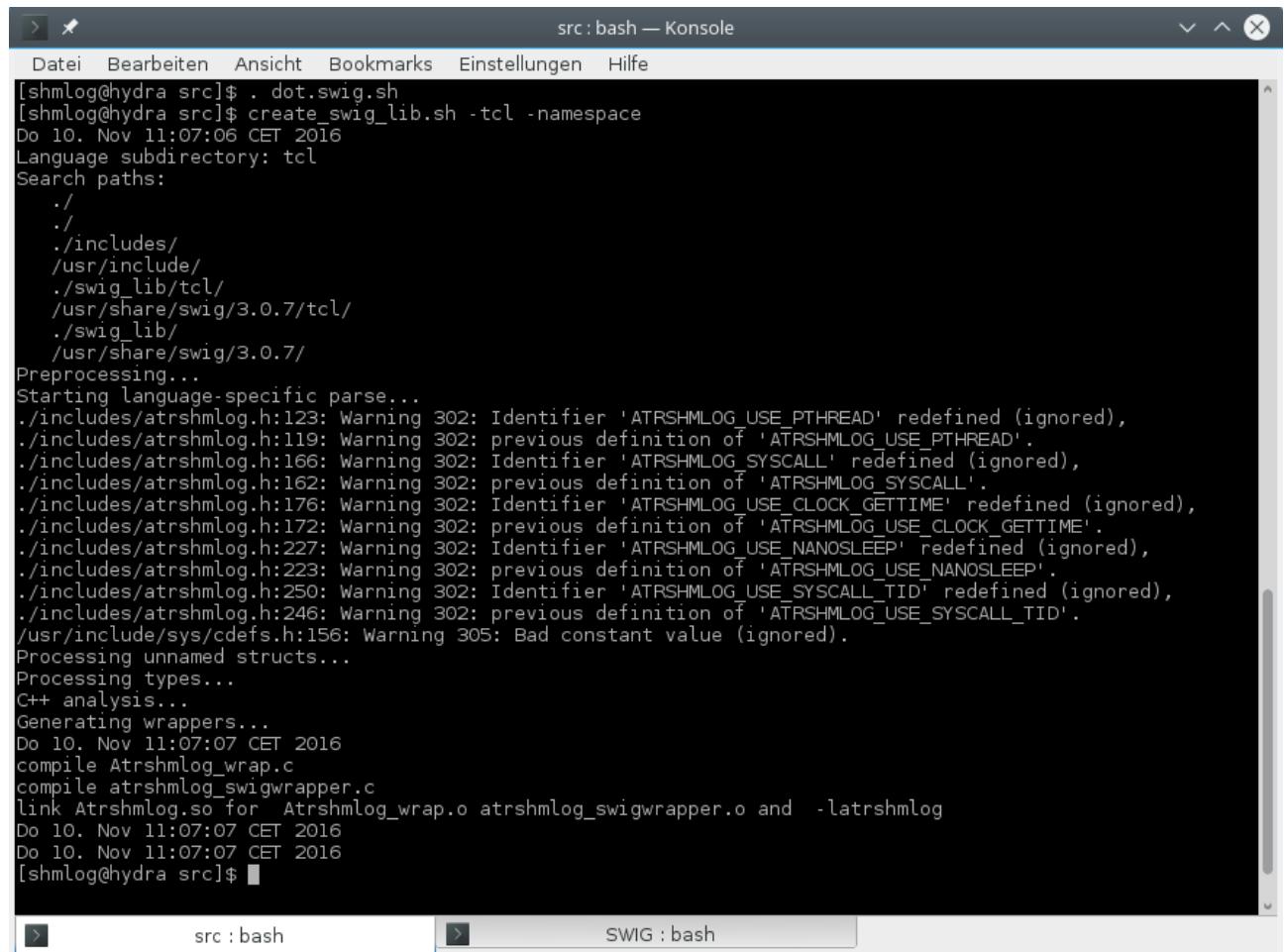
*Illustration 66: Setting the build environment*

No noise here. If you insist you can check the environment.

Next is to start the build.

## **Building with *create\_swig\_lib.sh***

We start the script. This time we don't have the target in place, so we have to give it as a parameter. And because the tcl does not have a module by default we give a second parameter to make it to surround our functions in a package with -namespace



The screenshot shows a terminal window titled "src : bash — Konsole". The command ". dot.swig.sh" is run, followed by "create\_swig\_lib.sh -tcl -namespace". The output shows the script determining the language subdirectory as "tcl", listing search paths including the current directory and standard include directories, and performing preprocessing and type processing. It then generates wrappers, compiles atrshmlog\_wrap.c, and links Atrshmlog.so. The process is completed with a final link step.

```
[shmlog@hydra src]$ . dot.swig.sh
[shmlog@hydra src]$ create_swig_lib.sh -tcl -namespace
Do 10. Nov 11:07:06 CET 2016
Language subdirectory: tcl
Search paths:
./
./includes/
/usr/include/
./swig_lib/tcl/
/usr/share/swig/3.0.7/tcl/
./swig_lib/
/usr/share/swig/3.0.7/
Preprocessing...
Starting language-specific parse...
./includes/atrshmlog.h:123: Warning 302: Identifier 'ATRSHMLOG_USE_PTHREAD' redefined (ignored),
./includes/atrshmlog.h:119: Warning 302: previous definition of 'ATRSHMLOG_USE_PTHREAD'.
./includes/atrshmlog.h:166: Warning 302: Identifier 'ATRSHMLOG_SYSCALL' redefined (ignored),
./includes/atrshmlog.h:162: Warning 302: previous definition of 'ATRSHMLOG_SYSCALL'.
./includes/atrshmlog.h:176: Warning 302: Identifier 'ATRSHMLOG_USE_CLOCK_GETTIME' redefined (ignored),
./includes/atrshmlog.h:172: Warning 302: previous definition of 'ATRSHMLOG_USE_CLOCK_GETTIME'.
./includes/atrshmlog.h:227: Warning 302: Identifier 'ATRSHMLOG_USE_NANOSLEEP' redefined (ignored),
./includes/atrshmlog.h:223: Warning 302: previous definition of 'ATRSHMLOG_USE_NANOSLEEP'.
./includes/atrshmlog.h:250: Warning 302: Identifier 'ATRSHMLOG_USE_SYSCALL_TID' redefined (ignored),
./includes/atrshmlog.h:246: Warning 302: previous definition of 'ATRSHMLOG_USE_SYSCALL_TID'.
/usr/include/sys/cdefs.h:156: Warning 305: Bad constant value (ignored).
Processing unnamed structs...
Processing types...
C++ analysis...
Generating wrappers...
Do 10. Nov 11:07:07 CET 2016
compile Atrshmlog_wrap.c
compile atrshmlog_swigwrapper.c
link Atrshmlog.so for Atrshmlog_wrap.o atrshmlog_swigwrapper.o and -latrshmlog
Do 10. Nov 11:07:07 CET 2016
Do 10. Nov 11:07:07 CET 2016
[shmlog@hydra src]$
```

*Illustration 67: Create the tcl library with SWIG*

And there we are.

First it starts the SWIG generator for a tcl module. After the swig is done its compile time.

One compile for the atrshmlog\_wrap.c .

Next compile for the atrshmlog\_swigwrapper.c .

And one link line.

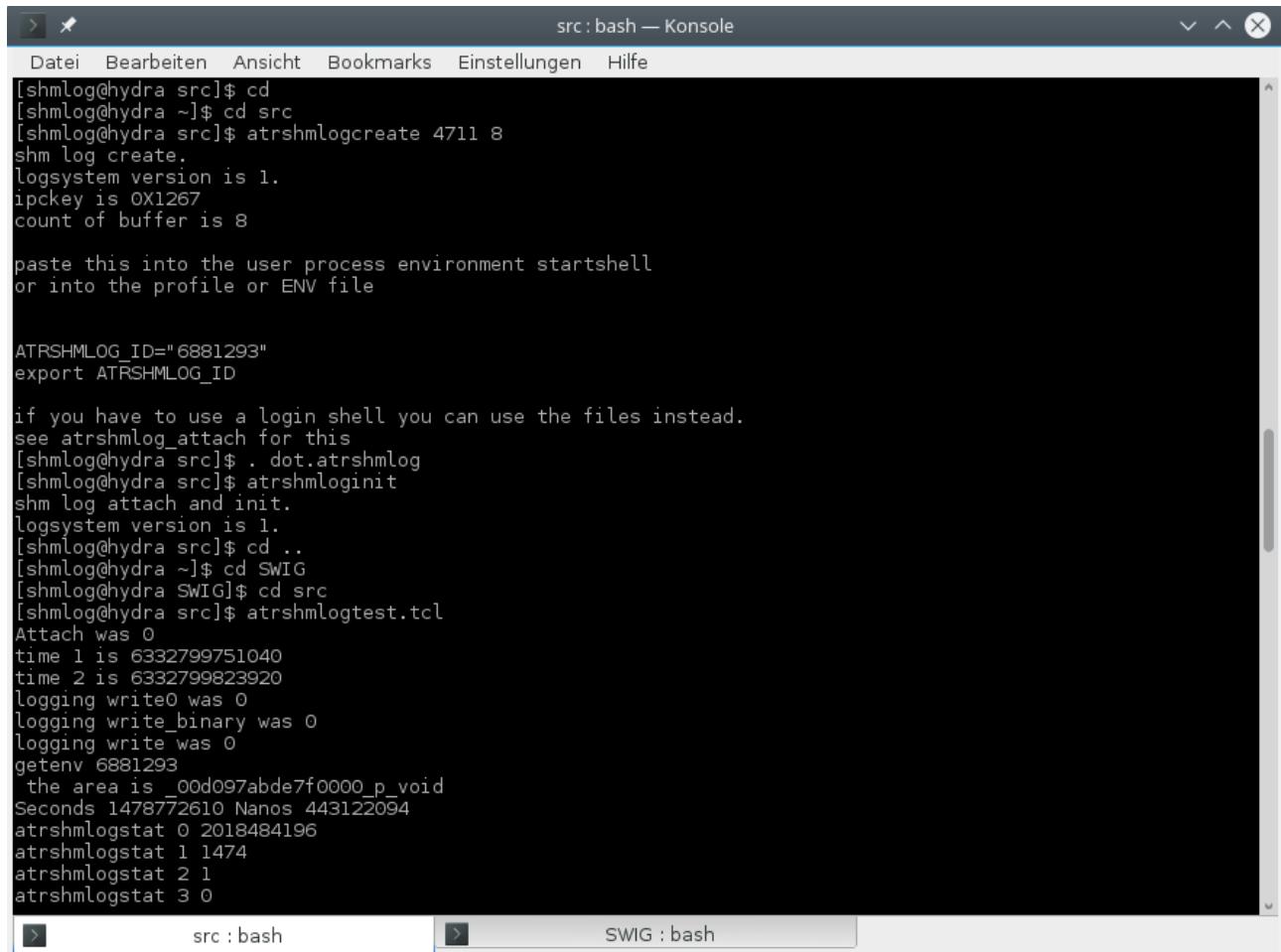
All what's left is the test.

## **Testing the tcl bridge**

We have first to create a shared memory buffer with atrshmlogcreate. I will use 4711 and 8 for it. Then the init for the area. Next the test of the bridge. Then the reader and the convert.

At last we can check for the result log.

And because it was already done for the C test program we do it in short form now.



The screenshot shows a terminal window titled "src : bash — Konsole". The window contains a command-line session. The user runs "attrshmlogcreate 4711 8" to create a shared memory buffer. It then runs ". dot.attrshmlog" to initialize the log. The log system version is 1, and the ipckey is 0x1267. The count of buffer is 8. The user is instructed to paste this into the user process environment startshell or into the profile or ENV file. The environment variable ATRSHMLOG\_ID is set to "6881293". The user then runs "attrshmlogtest.tcl" which attaches to the log, writes to it, and reads from it. The log area is at address 00d097abde7f0000. The log contains two entries with timestamps of 1478772610 and 1478772610 Nanos 443122094. The log is then converted back to binary using "attrshmlogstat". The log is successfully converted and read back.

```
[shmlog@hydra src]$ cd
[shmlog@hydra ~]$ cd src
[shmlog@hydra src]$ attrshmlogcreate 4711 8
shm log create.
logsystem version is 1.
ipckey is 0x1267
count of buffer is 8

paste this into the user process environment startshell
or into the profile or ENV file

ATRSHMLOG_ID="6881293"
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.
see attrshmlog_attach for this
[shmlog@hydra src]$ . dot.attrshmlog
[shmlog@hydra src]$ attrshmloginit
shm log attach and init.
logsystem version is 1.
[shmlog@hydra src]$ cd ..
[shmlog@hydra ~]$ cd SWIG
[shmlog@hydra SWIG]$ cd src
[shmlog@hydra src]$ attrshmlogtest.tcl
Attach was 0
time 1 is 6332799751040
time 2 is 6332799823920
logging write0 was 0
logging write_binary was 0
logging write was 0
getenv 6881293
the area is _00d097abde7f0000_p_void
Seconds 1478772610 Nanos 443122094
attrshmlogstat 0 2018484196
attrshmlogstat 1 1474
attrshmlogstat 2 1
attrshmlogstat 3 0
```

*Illustration 68: Test the new tcl library*

HM. Worked as expected.

And here the result for the log after the fetch from the reader.

The screenshot shows a terminal window titled "src : bash — Konsole". The window contains a log of command-line interactions. The user runs "atrshmlogstat" multiple times, followed by "cd" commands to change directories. Then, they run "atrshmlogstopreader" to attach and set a reader flag and pid. The logsystem version is 1. The user then runs "atrshmlogreaderd d1" to attach and loop write file. The logsystem version is 1. The directory is d1, files\_per\_dir is 10000, and stop via signal reader for pid 9999 with value 9999. The user creates a directory "d1" if it doesn't exist. The count of initial used fetchers is 4, and the count of initial used writes is 12. Logging is done. Writer data is shown with time 1584480, count 1, per use 1584480. An "atrshmlogconv d1" command is run to convert from file "d1/0/atrshmlog\_p5040\_t5040\_s0\_f0.bin" to file "d1/0/atrshmlog\_p5040\_t5040\_s0\_f0.txt". The file contains several lines of binary data representing log entries. The last command shown is "atrshmlogconv d1".

```
atrshmlogstat 94 0
atrshmlogstat 95 0
atrshmlogstat 96 0
atrshmlogstat 97 0
atrshmlogstat 98 0
atrshmlogstat 99 0
[shmlog@hydra src]$
[shmlog@hydra src]$ cd
[shmlog@hydra ~]$ cd src
[shmlog@hydra src]$ atrshmlogstopreader
shm log attach and set reader flag and pid.
logsystem version is 1.
pid before 0
flag before 0
[shmlog@hydra src]$ atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
logging done.
writer data : time 1584480 count 1 per use 1584480
[shmlog@hydra src]$ atrshmlogconv d1
shm log converter from file 'd1/0/atrshmlog_p5040_t5040_s0_f0.bin' to file 'd1/0/atrshmlog_p5040_t5040_s0_f0.txt'.
id 1 acquiretime 810 pid 5040 tid 5040 slavetime 1670 readertime 8860 payloadszie
117 shmbuffer 0 filenumber 0 sequence 0
[shmlog@hydra src]$ cat d1/0/atrshmlog_p5040_t5040_s0_f0.txt
0000005040 0000000000005040 000 0000000000000000 000006332799751040 000006332799823920 00000000000000072880
1478772610442959509 1478772610442986831 0000000000000000 000006332799751040 000006332799823920 00000000000000072880
0000005040 0000000000005040 000 0000000000000000 000006332799751040 000006332799823920 00000000000000072880
1478772610442959509 1478772610442986831 0000000000000000 000006332799751040 000006332799823920 00000000000000072880
0000005040 0000000000005040 000 0000000000000000 000006332799751040 000006332799823920 00000000000000072880
1478772610442959509 1478772610442986831 0000000000000000 000006332799751040 000006332799823920 00000000000000072880
hello, world
hello, anton world
[shmlog@hydra src]$
```

Illustration 69: The log result

OK. So we have the log in here.

## Details

Now we make it for the bridge in C.

Take your favorite text editor and open the Atrshmlog.i.

The file is not so complicated. Its only a bit bigger than a good SWIG example because of the rough hundred functions we have in here.

It starts with the usual comment stuff.

Then the definition of the module. I use a Camel case here like in most modules in perl.

Next is the code block for the wrapper file with the usual include of the C module.

What follows are the prototypes of the helpers that we will need.

There is only small need for helpers here – compared to the python its a really thin layer.

We need the otherwise inlined gettime.

And for the SWIG way to handle output the helpers of get\_inittime, get\_readtime and get\_statistics.

The SWIG way is done by conversion of output pointer parameters into temporary variables – and then to give back them on the interpreter in the natural way. For perl its an array on stack which makes them a simple return array. For tcl its a list.

So we get an tcl list with two values from the get\_inittime and the get\_realtime.

For the get\_statistics its a list of – well – hundred values for now. I simply say that's OK for me, after all I have hundred counters in place. And only 85 are in use now.

Last are the read and read\_fetch helpers. They need a buffer and I have made it a full length one.

So with the small hacks for peek and poke the included code finish.

Next is the typemap include. I had a strange result when I had a comment after the thing – so I removed the comment and anything is now back to normal.

Its for the OUTPUT parameters, see later.

Then we have the typemap for the c string buffer handling in read and read\_fetch.

I had some trouble with two things in the first place, so I added two defines to make the compiler work again.

Perhaps this is not needed for your SWIG implementation.

Then the ignore block follows. I simply cannot use the raw prototypes, so I use the include of the interface and the raw from the internal to make the SWIG see my C module functions.

This has a downside : I had to make for some variables a SWIG directive to NOT get some silly setter stuff.

And I catched some stuff from stdint.h here in too.

So that for the ignore block.

Next the translation of the names follow. Most is to cut of the atrshmlog\_ prefix. For the helpers its the additional atratrshmlog\_ prefix.

After that is set the include of the interface header is done.

Last are the prototypes of the helpers and the modifications to make the OUTPUT and cstring buffer things to work.

Now we can switch to the wrapper helpers code.

Open the atrshmlog\_swigwrapper.c with your favorite text editor.

We start with the usual comment stuff.

Then the includes. We need a strlen in the write, so we have also the string.h header here.

First is gettime. Its more or less the Macro version here.

Next is write binary. We get at least a address and the length in here.

Then the heart of the client log, the write. We use a string here and take its length. No binary in here. If you need binary use the write\_binary instead.

Next are the two getter for times. We need to make the output thing work, so no struct as return value in the SWIG layer. Instead we get with the OUTPUT typemap thing here two values on the return stack – which makes a array with two values for me. Perfect.

Then the get statistics. Again its only about a different interface and again the OUTPUT is used. This time you get 100 values in the return array.

For read and read\_fetch we use a thread local buffer – its a malloced one and a big one.

So if you really plan to do the read or read\_fetch thing you will need memory, and if you use multiple threads its for every thread an additional Mib.

We already made it for peek and poke in the definition file. So no implementation here.

And this is it. We are through alrea.... Ahem. OK.

The SWIG generated too.

I started after the XS didn't make it.

So first thing first – get an example and see how its done.

Swig's home page is OK for it. And it was a very simple one.

I checked also some things on google and found this should be easy. So I made the first approach.

And - BANG. Got compile errors for things I did not use in the first place – that's frustrating, crashing because of a thing you didn't need.

First was the conversion of a wchar\_t value – the generated function was simply not there.

So because I even don't need it I made a define hack in the compile script.

Next was a missing off64\_t. OK, next define.

After an hour I was up and running.

Next was to get info about the get\_statistics thing. I got a scalar reference – nothing you can use in perl for the thing.

I checked google and found a nice article for perl98 about a return of arrays in C for perl. The thing was easy to make, some simple helper to alloc mem, then do the call of get\_statistics with the returned scalar reference, then use some getter function to do the fetches into an perl array – and then delete the no longer allocoed in between array.

Sounds great – for a C developer.

For a perl user after 20 years, well, horrible. Guess its the same for the tcl folks.

So I checked again when I made the get\_inittime. And this time I found the real thing. A function that uses an OUTPUT in the prototype delivers a value back. Having more than one OUTPUT simply delivers multiple values. And for the types I had it was the simple number stuff – and so I got an array with two values back.

That was for a C developer a bit frustrating, but a helper did the job.

For the perl user it was - well, as expected. No big hooray or so. Simple use it. Same for my first steps with tcl lists.

Back to get\_statistics.

Make a helper, use the array in between and see the thing was done in the wrap.c again. And it was OK.

Nice. Now the last three hours – the read and read\_fetch.

Again the helper was needed because of the stuff that is delivered via pointers to the caller. I needed the OUTPUT again.

For the major part, the buffer, I had again to check. And I found a thing that worked for binary and did the allocation and the free in perl itself. OK, this was what I had in mind. For python the code was done in the core functions, so I simply made copy's and changed parts of them. After the three hours I had a working read\_fetch demo.

Last hour. Cleanup the no longer needed. Make some helpful constants and rename the things.

Again the help of SWIG.org did the job.

And so we have now a SWIG generated wrapper.

Check the doswig.sh for the conversion and for a small hack for a erroneous construct that I made here.

Now open the Attrshmlog\_wrap.c in your favorite text editor.

You find at the beginning the code for my two problematic compiler no-go.

Then you can skip the swig machinery code.

It about line 2050 when we are back in the module.

The SWIG first handles some variables and return struct things. I don't need them, but SWIG thinks it could not hurt.

For the variable's I tolerate the getter's. But the setters - I don't need them. So you are having now some immutable directives in the interface header to block that.

After the setters and getter's we enter the REAL stuff in 2250.

Attach.

Its a simple one, no parameters, only a return value.

And yes, its actual an int in Linux land here. So we get a simple wrapper.

Ignore the get\_statistics here, its dead.

Next is the write0 . Some more parameters, but that's it.

Sleep nanos is simple, only a parameter, and again its an int for SWIG and we are in.

As you can see it is no big deal to read it. But to write would be a lot more work. So I am happy the generator works....

After the environment handling stuff with strings we enter the big number of simple setter and getter like functions.

Most look like the first two of them.

Some minor things to mention.

The area things are done in the jni and python by using numbers. Here they are scalar references in perl, pointers in text in tcl.

About line 3750 we hit the otherwise inlined functions from the interface .

What follows are the prototypes from the definition file for the create, the delete, cleanup\_locks and init\_shm\_log. After this we have the helpers in place.

The gettime is easy. The write\_binary has some more to do. But again only parameter stuff.

With the get\_inittime we enter the magic of OUTPUT. It simply works, so I don't question it.

Then the real get\_statistics is in.

Its a big – but otherwise boring – version of the get time things.

And that's it for the wrap.c

## **Another platform : CentOS**

OK. My box is running a fedora 23, so no problem for the C 11 compiler.

But when I tried the CentOS 7.2 (at this time the best known clone for the RedHat 7 ) I found the compiler a bid off topic ( its now October 2016 and still a gcc 4.8 in place).

Yes, it had the c11 switch.

And yes, it had not the optional header stdatomic.h in place. So I googled a bit and came up with this.

How to make a running module for CentOS 7.2 ?

Its in the unsupported directory now. I will not integrate it because you can have a different compiler, so I leave that to you.

But I give you here the road map to do it.

## **Get the compiler to work that you need**

First problem for the compiler is that you can not get any rpm so far. Seems that the need is not that big after all, so you simply have to do it for the old way. Get the source and then build it from scratch.

Getting the source is not a problem. Entry point is [gcc.gnu.org](http://gcc.gnu.org). There we find links for downloads and we are in a tree that contains the whole versions available.

OK. We had a 4.8, and a 4.9 could do the job after some checks. But then I had experience with the 5.3 and so I switched to the latest 5.4 .

In theory you can make it local, no need for root here. But I still use root when it comes to a compiler so I made a g5 directory for root and unpacked the thing.

You should be careful before you start – its a 5 GB thing after all to make it.

Next I went into the configure

```
./configure --with-system-zlib --disable-multilib --enable-languages=c,c++
```

and got missing lib's for gmp, mpfr and mpc stuff.

HM. Had already made a full update and after checking with yum the things were already in place. So the devel companions where missing. Some yum install mpr-devel etc stuff ....

```
yum install gmp-devel  
yum install mpfr-devel
```

```
yum install *mpc*
```

did the work and I could end the configure successful .

Now the usual make... took a while, about 2 hours on my box.

Make install again .... done.

I found my new buddy at /usr/local/bin and it was x86\_64-unknown-linux-gnu-gcc-5.4.0.

So much for names you can expect today.

## Changing the build scripts

I started with bin then. First the compiler stuff.

```
case $ATRSHMLOG_PLATFORM in
    linux)
        # linux x86_64 gnu
        # CC="gcc -std=gnu11"
        CC="x86_64-unknown-linux-gnu-gcc-5.4.0 -std=gnu11"
        PICFLAG=-fPIC
        OPTMODE=-O3
;;
    ;;
```

That wasn't hard after all. Simply replace vanilla gcc. This was the g99.sh.

```
case $ATRSHMLOG_PLATFORM in
    linux)
        # linux x86_64 gnu
        # CC="gcc -pthread"
        CC="x86_64-unknown-linux-gnu-gcc-5.4.0 -pthread"
        LIBMODULE=-latrshmlog
;;
    ;;
```

and this was ell.sh.

For the dot file I made a copy dot.platform.sh.centos and adapted.

```
date
```

```
echo "set basedir "
# we set the basedir. so this must be used inplace in the correct dir
ATRSHMLOG_BASEDIR="$(pwd)"

export ATRSHMLOG_BASEDIR
```

```
echo "set platform"  
# we set the platform for build  
  
ATRSHMLOG_PLATFORM=linux  
  
# ATRSHMLOG_PLATFORM=cygwin  
  
# ATRSHMLOG_PLATFORM=mingw  
  
export ATRSHMLOG_PLATFORM  
  
echo "set path"  
# we need a working path. so we assume its the basedir relative in bin  
# what makes the thing
```

```
PATH="$PATH:$ATRSHMLOG_BASEDIR/bin:..:/usr/local/bin"
```

```
export PATH
```

```
date
```

```
# end of file
```

Not much difference, only the additional /usr/local/bin in the PATH.

The makeall.sh ran till it came to the C++ files. There I had at least some luck, the 4.8 could have made it if it weren't for the multithreading. It simply could not make the C++14 switch, so I had switched back to c++11, but that didn't work either. It complained to need a new one. So I gave it to the thing in g++14w.sh.

```
case $ATRSHMLOG_PLATFORM in
```

```
    linux)
```

```
        # linux x86_64 gnu
```

```
#      CPP="g++ -std=c++11 -pthread -Wall -Weffc++ -fdump-tree-original"
CPP="g++ -std=gnu++1y -pthread -Wall -Weffc++ -fdump-tree-original"
PICFLAG=-fPIC
OPTMODE=-O3
LIBMODULE=-latrshmlog
;;

```

After that my makeall.sh ran complete.

In the end I integrated that in to the platform script for centos and now you only have to source that platform dot file and with the compiler in place and the rest will use that compiler.

## Testing

And that's it. I made a short test run and got my wanted result:

```
0000009300 0000000000009300 000 000000000000000000 000086453358488444  
000086453358488444 000000000000000000 1477592391142288024 1477592391142288024  
000000000000000000 0000000001 P 0000000001 hello, world.
```

So there is nothing special for the CentOS 7.2, its only the compilers that had a glitch for me this time. The rest works as expected. I assume its the same for all from 6.0 on up to today's 7.2.

If the RedHat team will switch to a newer version of gcc you can forget this. Its only an interim needed workaround, and if you can get a rpm even not the time consuming compile is needed.

If you have to do it you must have the library's in place and you must have sufficient space for it. In principle you can say its a needed precondition for the module and you are through. And if you don't install that thing global, but in a local directory of yours you even don't have to be root for it.

So I can live with that.

## **Another platform : FreeBSD**

So after making it for cygwin, mingw and another Linux it was time to try another posix platform. I had checked for the BSD UNIX's and found that FreeBSD was the most popular one.

So I made an installation on my virtualbox of the 11 release.

I am a newbie on the BSD, so I thought it should be different to Linux, but not so far away as a mingw.

After the install was done I found I was wrong about not so far from Linux. It was like being back in the RedHat 6 installation of 1999 – and yes, I mean the Red Hat 6, not the RHEL 6 here.

All I had was a vga terminal based installation – OK, I can live with that.

Next I found that I had to install even the package installer itself – I had made all the tiny switches and I had a 2.5 GB dvd image, but still a basics only installation – so much for using a big download image...

Next came that thing I call working environment was not there. I wanted to get a GUI environment running.

After one day I stopped that. I had done a lot of installs, but even with the ports thing I could not get it done right – having a KDE4 with US EN only is a bit sadistic when you have a german keyboard on your notebook....

So I installed the emacs, made login via ssh -X and worked in the ssh session instead of the GUI.

After that the thing was done in about 3 hours.

I had to switch to another bash place in the scripts and for perl. But that was not the only thing, also I had to change the scripts to accept a new platform – bsd – and later on a new flavour – freebsd.

When openbsd came in I switched to numbers and after one glitch the freebsd became the flavour 3 and openbsd the flavour 4.

I don't plan to make flavour's of mingw and cygwin, so they have a 1 for now.

After checking for the compiler I found a gcc from one of my installations – the bsd guys try NOT to have the newest gcc because of license problems. So the freebsd was in fact based on clang and not on gcc. But if you install anything serious you have one back ...

Perfect. A new platform and a new compiler.

After some googleling I found I could make it with clang for the inline code assembler. GOOD.

The check system became a new part, the bsd and clang were added and I found in the end only one thing strange. The call to get the thread id was for every bsd different. So I added the new platform in rough 2 hours and then made it through the compile.

The test was easy and I had my new platform up and running.

The rest is in the unsupported directory. Check for freebsd.

The stuff has been integrated now into the build scripts, but for the ksh support you will have to edit them in this release.

## **Another platform : OpenBSD**

So after making it for cygwin, mingw and another Linux it was time to try another posix platform. I had checked for the BSD UNIX's and found that FreeBSD was the most popular one.

So I made an installation on my virtualbox of the 11 release.

The next best was, according to an Wikipedia entry of 2005, OpenBSD.

When I had made the FreeBSD I had already encountered a feeling of rough said old time – but OpenBSD seemed to top that in every aspect.

The compiler was still a gcc 4.2 – so no chance to do it with this one.

I found a 4.9.3 as optional package – OK, I thought I was through then.

And I was wrong. This 4.9.3 did NOT support the stdatomic.h . BAD.

I checked some Google entries and started to try to make it for my own gcc.

First I tried the 5.4.0 from my centos ride.

No luck – the build did not start at all.

After identifying the missing headers of gmp, mpfr and mpc I added the Usr/local and at least my configure was ok now.

Then I had to install gmake – the make was not capable to use the Makefile. That's OK for me, I know why I don't use make in here after all ...

So now the tools were doing fine.

But not the Build. I crashed in some gomp lib thing.

Did I need this thing ? – NO.

So I tried to make it without it. But no luck – no switch for setting OFF the GOMP...

HM. After doing it with a 4.9.4 and crash landing the same way I found the missing golden thing - -fPIC did the thing ....

Next moment I crash landed in the build for stdc++ ...

OK. Start over, this time I only wanted the c language.

AgainI crash landed in the lib building stdc++ ... so much for I don't get what I don't want in GNU gcc....

I ended by stopping my gcc experiments and switching over to clang.

This was because it was not there in the first place – the clang package is llvm, and so my tries with a clang didn't work first – but when I had found the hint on the web that is was in from september on I found it after some minutes.

OK. So I had already made it for FreeBSD with a clang – so what could ...

BUMM. Again I crash landed.

This time it was a simple “no support for thread local storage” in my very first C source ...

HM. This was a new one. I had a relative new clang – 3.8.2 for freebsd, 3.8.0 for openbsd – and still that different behaviour. WOW.

After some checking I found no alternatives for the compiler, no magic switch to make it do the thing. It was simply a platform problem. Googeling made me clear that I could not have it this way.

So my search focused on alternatives. And I did find at least a dozen links for something that could do the job – but every time I investigated I ended up with a “...but it is no way to replace the on openbsd missing support for ...” thing. HAHA.

OK. Drastic problems need sometimes drastic meatures.

So I checked the pthread for a solution. And I found the specific data functions.

After a test run I new what to do: switch over to a flavour in the bsd clang parts, use thread local where it is possible, and make a new approach to the pthread specific for the one system that didn't made it otherwise.

After 15 minutes the thing worked and I started the backport.

This was the moment when I got some mess in my headers for the macros. It was clear that I had now to make use of the flavour also in my defines and so I switched from names to numbers. Looking BAD, but working GOOD.

After that I made some additional error handling – till that moment I had one function that deliverd my thread local, now it could deliver not only thread local but also thread specific, but also a NULL pointer was now in.

After making the error enums bigger and the checkings I ended up with two more hours.

Then I made the final build for openbsd again, now with the new top of my master branch.

The I had still those annoying linker messages about strcat, strcpy and sprintf usage...

OK. Again three more hours of work and the openbsd was happy.

This still does not make a bit more safety, but the guys of that system seem to me like the old day guys of WWII , always arguing that a victory is not possible without a big fleet of battleships .... perhaps someone tells them that we have today carriers, drones and cruise missles for the job ...

## **Another platform : NetBSD**

So after making it for cygwin, mingw and another Linux it was time to try another posix platform. I had checked for the BSD UNIX's and found that FreeBSD was the most popular one.

So I made an installation on my virtualbox of the 11 release.

The next best was, according to an Wikipedia entry of 2005, OpenBSD.

After that thing I switched to the system that runs simply everywhere – from alpha to zaurus... the NetBSD.

The install was as already suspected a mix of commandline and vga gui boxes.

It was a 7.0.2 – for the download - but most parts identified itself still as 7.0.1.

After the thing was done the usual changes.

The sshd got a X11 forward, the root a login.

My next thing was to make pkg\_add running. Some googleing did the job.

I found a doxygen, then the emacs, then the gcc 6.2.

After this I copied the source and started the adjustments.

The new platform and flavour made it to the scripts and then to the headers.

For the code the adjustments were minor. Only the additional flavour made more work than needed.

Later I replaced the stuff at most places with a new define and handled the thing now in the header.

OK. The adjustment took rough two hours. The best new thing was the leight weight processes – they seem to like different names for these things on the BSD's.

After the first test worked I made the backport and after two hours I had it up again and now with the new define in place.

Copy of the binaries to the new netbsd in unsupported and the documentation.

All in all this was a 6 hour thing.

## **Another platform : Solaris**

After the BSD UNIX's it was time for the last platform of the established – Solaris.

The opensolaris 11.3 is the last edition of that former so vital platform.

Oracle simply does not develop any more, they focus on the Java and Database thing. So no more Solaris.

OK. There is a OpenIndiana now, but this project even does not support the sparc and sparc64 – so I think I have to pass that and start with the last Oracle version. Perhaps I go back to indiana next week – its a fast thing after all I got on Solaris.

Install was a mix of command line and some gui like boxes.

After I had made it, it took about 8 GB of discs – but still no cc, emacs or X support ...

Found a switzerland based project to support for it – the opencws.

So I installed that too, then got the emacs from there.

But no luck. Problems with X11 forward... OK, I had made the sshd\_config changes but still ...

Found the xauth was not there – and so the simple message was kind of right that xauth could not set the cookies ...

Switched to pkg and installed xauth. And with libxaw5 the thing came up.

BUMM. But it didn't work right – messy update of the thing – nothing you can really use to read a file. Founr something about font problem ... well, then I pass it.

I gave up. Installed the gcc – this time a 5.2 – from cws and switched to emacs with /ssh: mode...

After some checking I had the bin files in place.

Then the header and then the impls.

Compile did its job, and...

The atrshmlogcreate gave me a error 0 ?!??

After some checking of the man page I found no IPC\_... in Solaris beside CREATE .

OK, this was a minor thing. After all I have for the platform stuff for each platform the code separate, so a simple change made it.

After the test I started the backport.

It was rought 4 hours till I hit the doc now.

## **Another platform : Ubuntu**

After the BSD UNIX's and the Solaris it was time to finish the x86 64 platforms I had on my list.

So I started with Ubuntu.

The download was short- the server took only a CD, no DVD.

After the install I had to install the X stuff – apt install xterm did it.

Then the emacs, the doxygen and gcc. It came with a gcc 6 , so no problems.

After rough an hour I got my binaries.

So I added a platform, but it has no impact for now.

## **Another platform : OpenSUSE**

After the BSD UNIX's and the Solaris it was time to finish the x86 64 platforms I had on my list.

So I started with Ubuntu. Next was the OpenSuse. I had an somewhat outdated 12.2 and I stopped it.

From the download I got the leap 42.2 .

A full distro of rough 4 GB dvd ...

But no gcc that I could use.

So I installed the gcc-6 and gcc6-c+ after it.

A new flavour was the result for the new names of the compiler.

After that everything as usual.

Binaries in about an hour.

## **Another platform : Debian 8.6**

After the BSD UNIX's and the Solaris it was time to finish the x86 64 platforms I had on my list.

So I started with Ubuntu. Next was the OpenSuse.

Next the Debian.

I had luck – the 8.6 barely arrived and it came in with a 4.9.2 – gladly with stdatomic.h ....

So this was about an hour and I had the binaries.

A new flavour but I didn't need it.

## **Another platform : SLES**

After the BSD UNIX's and the Solaris it was time to finish the x86 64 platforms I had on my list.

So I started with Ubuntu. Next was the OpenSuse. Then the Debian.

And now SLES.

First things first. I started in 1994 with a SUSE 5.3 Distro my Linux experience. Then I switched 1995 to Caldera – even did it because of the new fancy Matrox Millenium support and I really go a 4 MB up and running, then invested about 400 DM in the upgrade to 8 MB and got a taste of a Red Hat based system with a fast desktop.

I have still up and running a Red Hat 2.1 and a Caldera on top of kvm – so much for my SUSE experience that time.

In 1999 I got more and more trouble with my C++ project Hexagon on the fenster;plural – and after waiting for 2 years for an ISO conformant library handling I gave up .

So it was time for Red Hat 6. Then 7, 8 and 9.

After this I switched to fedora – and today I still try this instead of the so called long term supported business Linux's.

For SLES this was always a handicap – I am a Red Hat guy....

Download of two DVD images, Install – well, was nice compared to the vga style systems before.

But the registration key didn't work – HM.

Then my first problem. How to switch it from DHCP to static when the bastard didn't support the video right, and so I even didn't see the panel for networking in this minimalistic 640x 480 desktop....

Next I tried the yast in terminal mode – but again no luck. This time the terminal catched my function keys, so there was no way to confirm my settings after switching to some wildly divergent panels... I hate that.

I switched to the ssh and to the terminal version there and then got my IP changed.

Next I tried to get a gcc for 5 or 6, but NO luck. The thing even didn't try the network repository because of my broken registration code – at least I assume that.

OK. Close your eyes, count to 0 from 10 backwards, a deep scream “scheisse” and I went back to work.

I took the gcc 5.4.0 source to the thing. Of course the usual problem with the gmp, mpfr and mpc – libs were in place, but no sign of the headers. GRRRR.

No way to install something from the 4.5 GB DVD that could help, too.

So I downloaded the gmp, mpfr and mpc libs, made the configure make make install thing and then

I could start my gcc 5.4.0 build.

OK, this is after all a Linux, and its important in the big business – the zOS machines and most of the inhouse servers that I know run on SLES and not on Red Hat in the companies I worked the last 4 years. So I wanted to support that whatever I have to do ....

After 6 hours of compiling I got my gcc 5.4.0.

After 10 minutes I had a set of binaries. But no doxygen – so I introduced a new flavour 10 and still had the doxygen in place – I hope you sles fellow can install that before you try to build. And that you don't need the sources to clean up that compiler mess.

Personal note:

In 1939 when WWII started the military guys had on top of their lists the big battleships with their 38+ cm main guns and the 28+ knots speed and the top class amour all around. So they insisted that it would be mandatory to get those ships in place and to attack. In May 1941, the last real battle took place and the at this time biggest military ship, the Hood, were sunk by Bismark and Prinz Eugen in about 6 minutes. The Prince of Wales was back to the construction set – nearly killed in one minute of cross fire.

Three days later the fall of the battleship began when the Ark Royal could made its move and torpedoized the Bismark with one of here Sword Fish bombers ( that's a over 15 years old doppeldecker thing made of mostly wood that tim, not a fancy spitfire or tempest).

The rest of WWII was the history of battleships sunk by plane.

Today I have the impression that some Linux distros try the old WWII way with the battleship : no actual software, compilers about 2 major numbers back behind actual development, libs outdated and no developer support. That's so “safe” and “well proven” ...

Sounds like the Ubuntu got its famous glory after all because it did not follow that path.

And so I can only pray that the Linux distros start to learn from history and evolve. There was a time when a Red Hat had a brand new and even unofficial 2.96 gcc on board – today I have to compile the 5.4.0 by myself to support a 6 year old ISO standard in C on the CentOS 7.2 which is a Red Hat Enterprose 7.2 in fact...

Is there anyone who feels like me that there is something wrong in the way of handling the factor “modern” in this business linux world ? Have we really gone that old and fossile in the linux business ? And have we really not seen the consequences ?

# Another platform : cygwin

For those who want to use it, there is a plain cygwin port.

Cygwin serves for the other port to fenster;plural as a build environment, but is not used for the module. See the mingw for that.

So the cygwin build is for those who are really trying to use cygwin – and there are at least some guys who are really interested in it.

So here we are for a cygwin.

You have to download not the full distro.

First check the cygwin master at

<http://www.cygwin.com>

for the latest download binary.

Load it on the box.

Then start it and give the permissions the thing needs.

After this you have to select some directory's and places and then the checking for download packages starts.

What to download ?

There is some stuff that is needed, but its only a small fraction.

- BASE

The base package of course. All in.

- Admin

The cygrunsrv. This is vital because you wont get shared memory if you are not having it up. This was formerly the cygserver and in BASE. Don't know why but now the script to configure is in BASE and its cygserver-config, but the main binary has been moved ( at least in the version i got in 10.2016).

- Devel

cygwin-devel, binutils, doxygen, gcc-core, gcc-g++, gdb (add the mingw if you plan to do that too)

- doc

## cygwin-doc

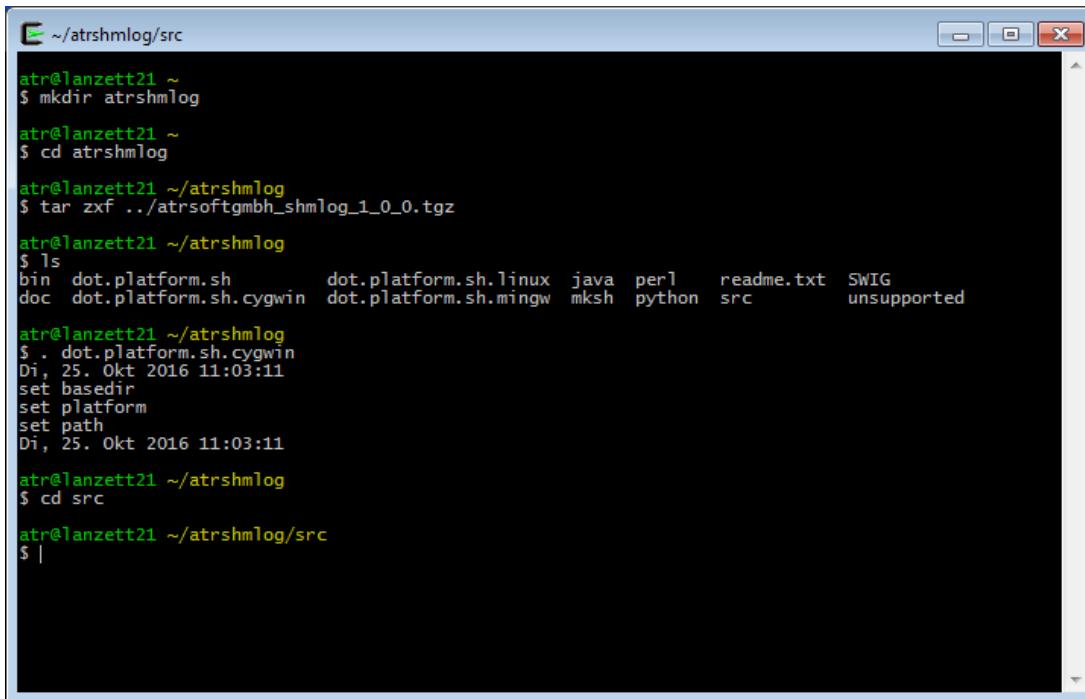
- editors
  - emacs, vim
- interpreters
  - perl
- libs
  - all in.
- shells
  - bash
- utils
  - dos2unix
- publishing
  - texinfo and doxygen

That should do the job.

The rest is still the 1.0.0 version – so much for my love for cygwin. You simply have to use the cygwin dot file and no copy of headers any more. The rest is simply the same (ok, there is the tests now, but I hope you can figure that out by yourself).

## After the unpack

We simply change to the BASEDIR



The screenshot shows a terminal window titled '~/atrshmlog/src'. The terminal output is as follows:

```
atr@lanzett21 ~
$ mkdir atrshmlog
atr@lanzett21 ~
$ cd atrshmlog
atr@lanzett21 ~/atrshmlog
$ tar zxf ../atrssoftgmbh_shmlog_1_0_0.tgz
atr@lanzett21 ~/atrshmlog
$ ls
bin  dot.platform.sh      dot.platform.sh.linux  java  perl    readme.txt  SWIG
doc  dot.platform.sh.cygwin  dot.platform.sh.mingw  mksh  python  src      unsupported
atr@lanzett21 ~/atrshmlog
$ . dot.platform.sh.cygwin
Di, 25. Okt 2016 11:03:11
set basedir
set platform
set path
Di, 25. Okt 2016 11:03:11
atr@lanzett21 ~/atrshmlog
$ cd src
atr@lanzett21 ~/atrshmlog/src
$ |
```

*Illustration 70: The cygwin base directory after unpack*

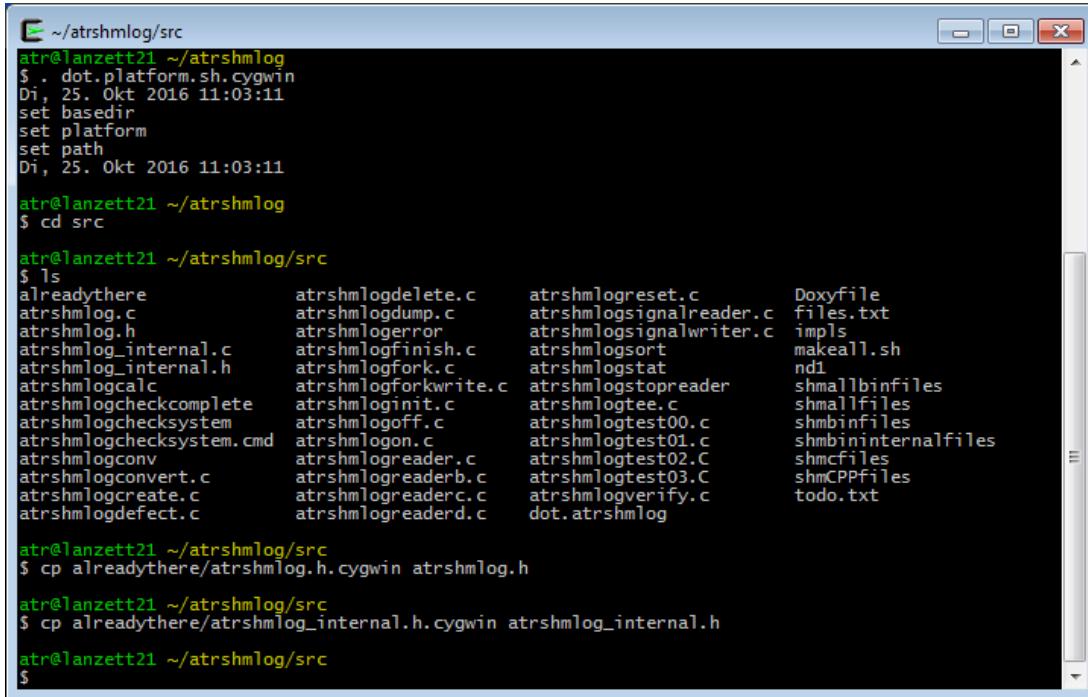
We start with source of the platform file. Its already there. Its name is

dot.platform.sh.cygwin

as you already could see in the picture.

## Prepare headers – NO MORE ....

Next we prepare the headers. They are already adapted so we only have to copy them from the alreadythere directory.



```
~/atrshmlog/src
atr@lanzett21 ~/atrshmlog
$ . dot.platform.sh.cygwin
Di, 25. Okt 2016 11:03:11
set basedir
set platform
set path
Di, 25. Okt 2016 11:03:11

atr@lanzett21 ~/atrshmlog
$ cd src

atr@lanzett21 ~/atrshmlog/src
$ ls
alreadythere    atrshmlogelete.c    atrshmlogreset.c    Doxyfile
atrshmlog.c     atrshmlogdump.c    atrshmlogsignalreader.c files.txt
atrshmlog.h     atrshmlogerror.c   atrshmlogsignalwriter.c impls
atrshmlog_internal.c atrshmlogfinish.c atrshmlogsort      makeall.sh
atrshmlog_internal.h atrshmlogfork.c  atrshmlogstat      nd1
atrshmlogcalc   atrshmlogforkwrite.c atrshmlogstopreader shmailbinfiles
atrshmlogcheckcomplete atrshmloginit.c  atrshmlogtee.c   shmailfiles
atrshmlogchecksystem atrshmlogoff.c   atrshmlogtest00.c  shmbinfiles
atrshmlogchecksystem.cmd atrshmlogon.c   atrshmlogtest01.c  shmbinternalfiles
atrshmlogconv    atrshmlogreader.c  atrshmlogtest02.c  shmcfiles
atrshmlogconvert.c atrshmlogreaderb.c atrshmlogtest03.c  shmCPPfiles
atrshmlogcreate.c atrshmlogreaderc.c atrshmlogverify.c todo.txt
atrshmlogdefect.c atrshmlogreaderd.c dot.atrshmlog

atr@lanzett21 ~/atrshmlog/src
$ cp alreadythere/atrshmlog.h.cygwin atrshmlog.h

atr@lanzett21 ~/atrshmlog/src
$ cp alreadythere/atrshmlog_internal.h.cygwin atrshmlog_internal.h

atr@lanzett21 ~/atrshmlog/src
$
```

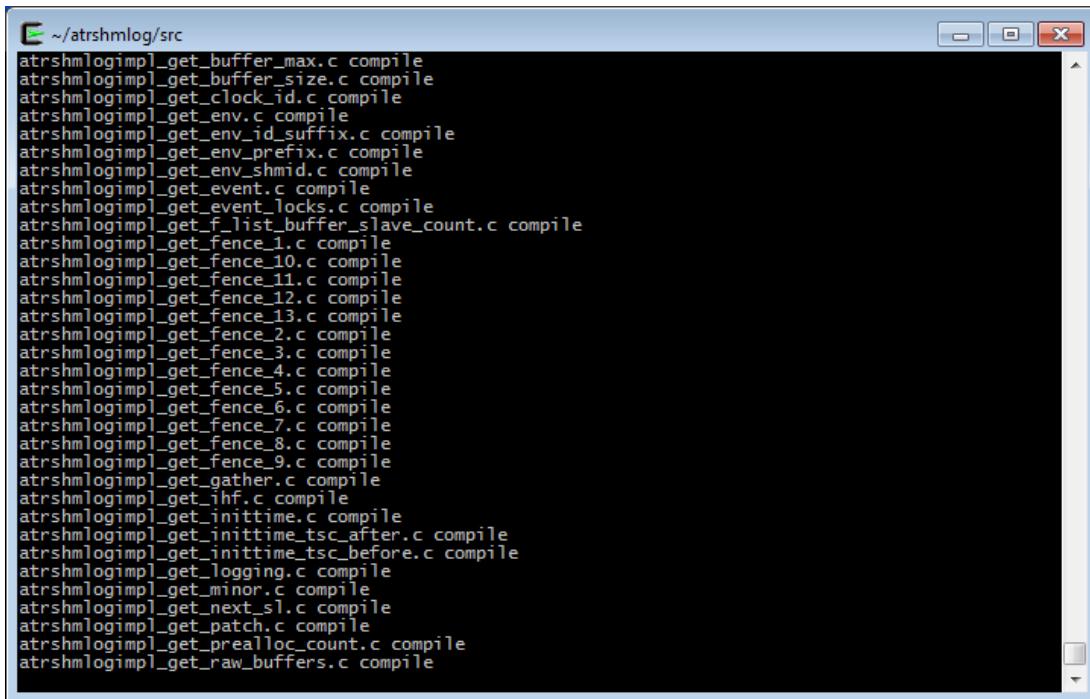
Illustration 71: Copy for the adapted headers.

Nothing dramatic so far. We start with the first compile.

FORGET THAT: now its done by the platform itself.

## First compile

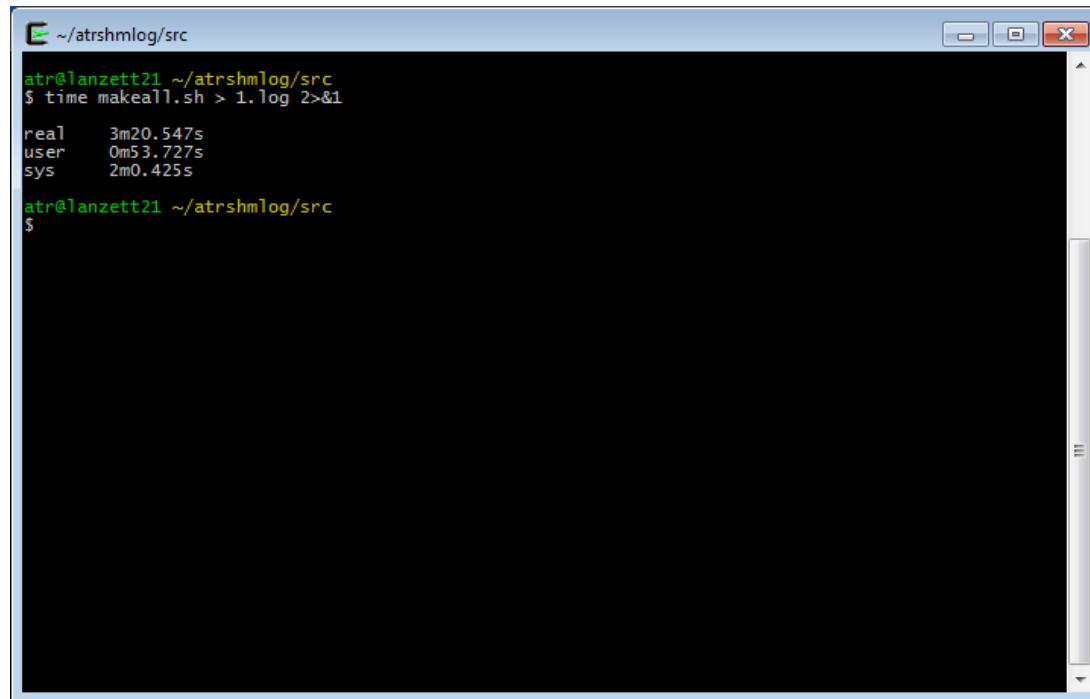
The makeall.sh in action:



```
~/atrshmlog/src
atrshmlogimpl_get_buffer_max.c compile
atrshmlogimpl_get_buffer_size.c compile
atrshmlogimpl_get_clock_id.c compile
atrshmlogimpl_get_env.c compile
atrshmlogimpl_get_env_id_suffix.c compile
atrshmlogimpl_get_env_prefix.c compile
atrshmlogimpl_get_env_shmid.c compile
atrshmlogimpl_get_event.c compile
atrshmlogimpl_get_event_locks.c compile
atrshmlogimpl_get_f_list_buffer_slave_count.c compile
atrshmlogimpl_get_fence_1.c compile
atrshmlogimpl_get_fence_10.c compile
atrshmlogimpl_get_fence_11.c compile
atrshmlogimpl_get_fence_12.c compile
atrshmlogimpl_get_fence_13.c compile
atrshmlogimpl_get_fence_2.c compile
atrshmlogimpl_get_fence_3.c compile
atrshmlogimpl_get_fence_4.c compile
atrshmlogimpl_get_fence_5.c compile
atrshmlogimpl_get_fence_6.c compile
atrshmlogimpl_get_fence_7.c compile
atrshmlogimpl_get_fence_8.c compile
atrshmlogimpl_get_fence_9.c compile
atrshmlogimpl_get_gather.c compile
atrshmlogimpl_get_lhf.c compile
atrshmlogimpl_get_inittime.c compile
atrshmlogimpl_get_inittime_tsc_after.c compile
atrshmlogimpl_get_inittime_tsc_before.c compile
atrshmlogimpl_get_logging.c compile
atrshmlogimpl_get_minor.c compile
atrshmlogimpl_get_next_sl.c compile
atrshmlogimpl_get_patch.c compile
atrshmlogimpl_get_prealoc_count.c compile
atrshmlogimpl_get_raw_buffers.c compile
```

*Illustration 72: Compile the module with makeall.sh*

After a full 3.5 minute round we have the result (that was before the t\_test.sh and tests ...).



The screenshot shows a terminal window titled '~/atrshmlog/src'. The command \$ time makeall.sh > 1.log 2>&1 is run. The output shows the execution time: real 3m20.547s, user 0m53.727s, sys 2m0.425s. The terminal window has a light blue border and a dark background.

```
~/atrshmlog/src
$ time makeall.sh > 1.log 2>&1
real    3m20.547s
user    0m53.727s
sys     2m0.425s
$
```

*Illustration 73: Time for a compile in cygwin on my box*

## The cygserver start

Next we need the cygserver to run. So we have to use an administrative shell this time:

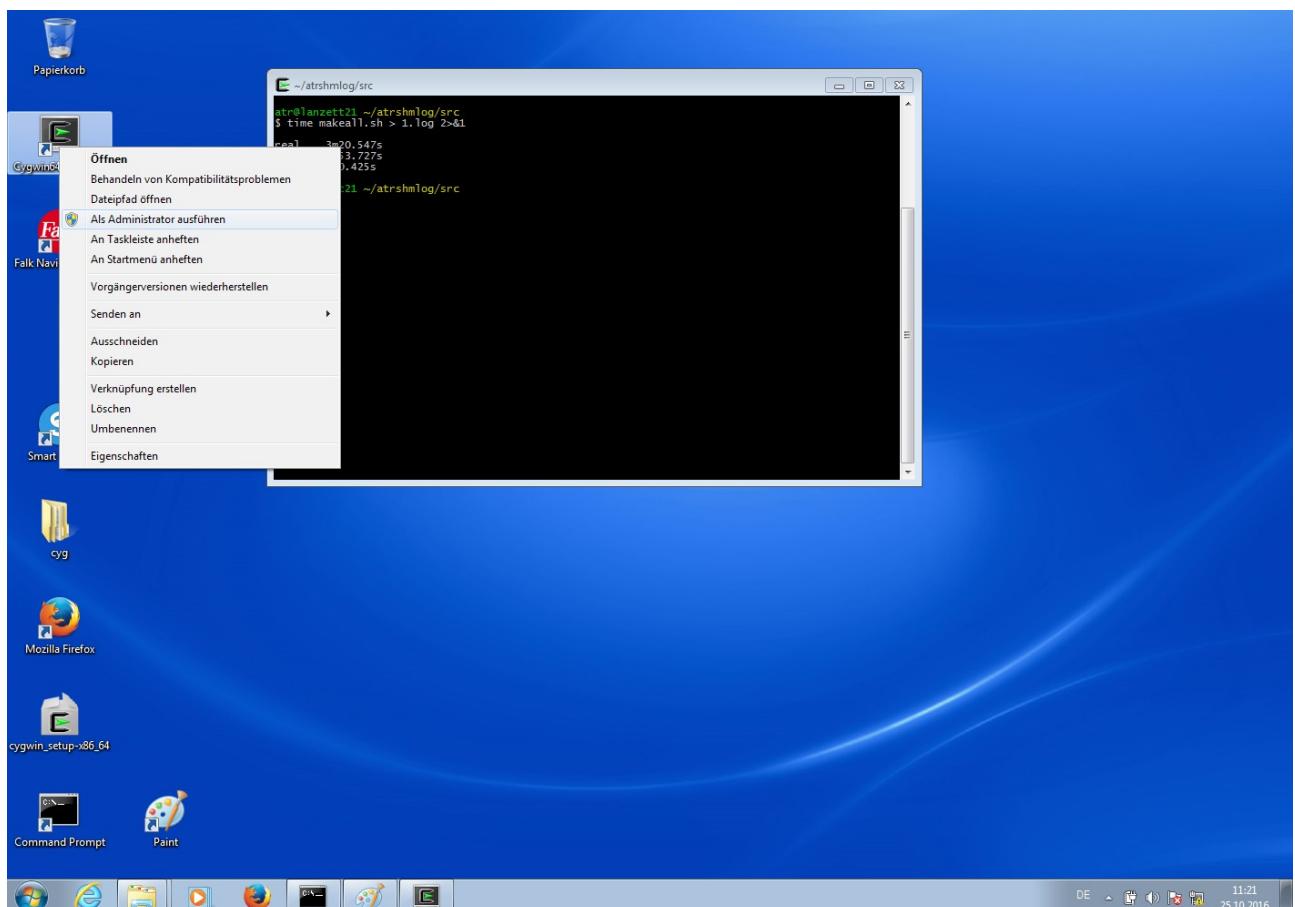


Illustration 74: Getting an administrator's shell

Open it with the right button context menu as administrative application.

OK. We can now start the server . Its mandatory for the use of the module. No ipc operations work without it.

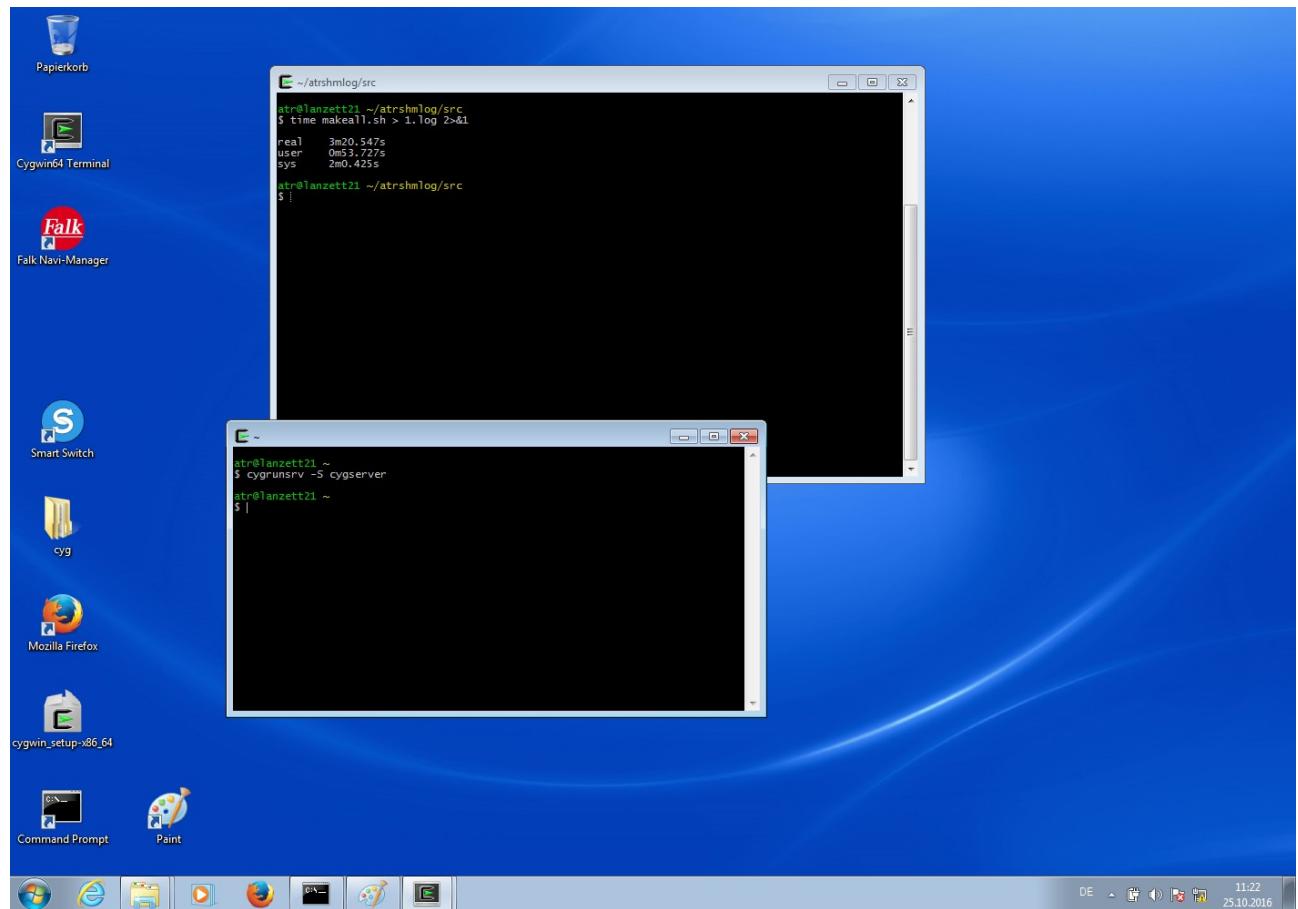


Illustration 75: Starting the cygrunsrv for the service cygserver

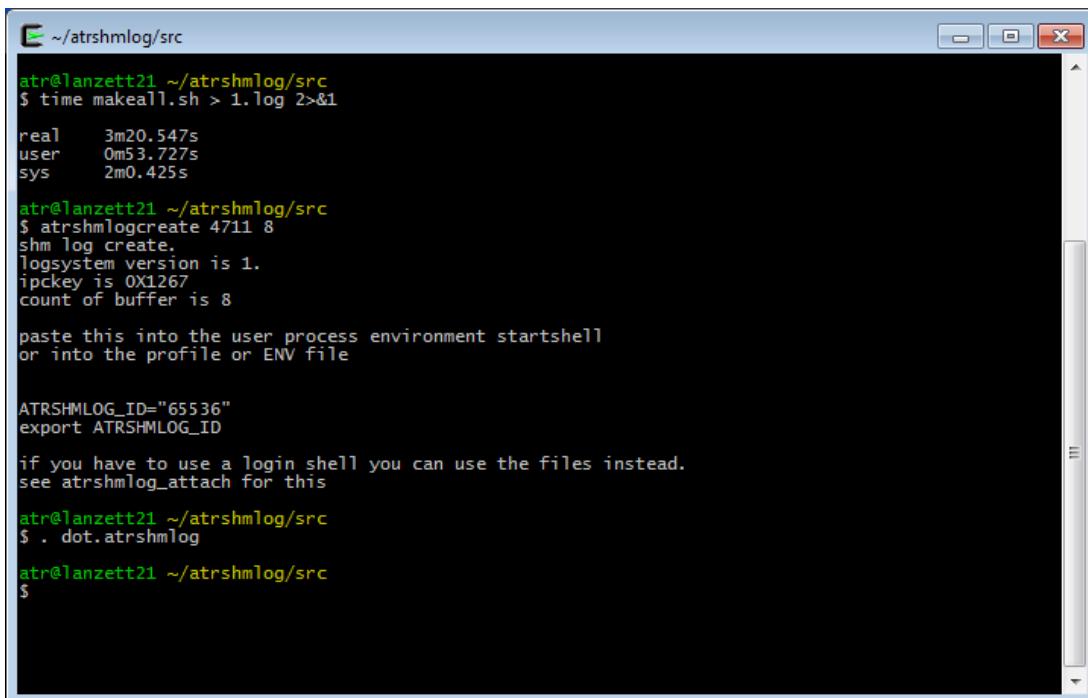
Up and running.

## Create of a buffer

Now we can create the buffer as usual.

```
$ atrshmlogcreate 4711 8
```

does the job.



The screenshot shows a terminal window with the following session:

```
~/atrshmlog/src
atr@lanzett21 ~/atrshmlog/src
$ time makeall.sh > 1.log 2>&1
real    3m20.547s
user    0m53.727s
sys     2m0.425s
atr@lanzett21 ~/atrshmlog/src
$ atrshmlogcreate 4711 8
shm log create.
logsystem version is 1.
ipckey is 0X1267
count of buffer is 8
paste this into the user process environment startshell
or into the profile or ENV file

ATRSHMLOG_ID="65536"
export ATRSHMLOG_ID

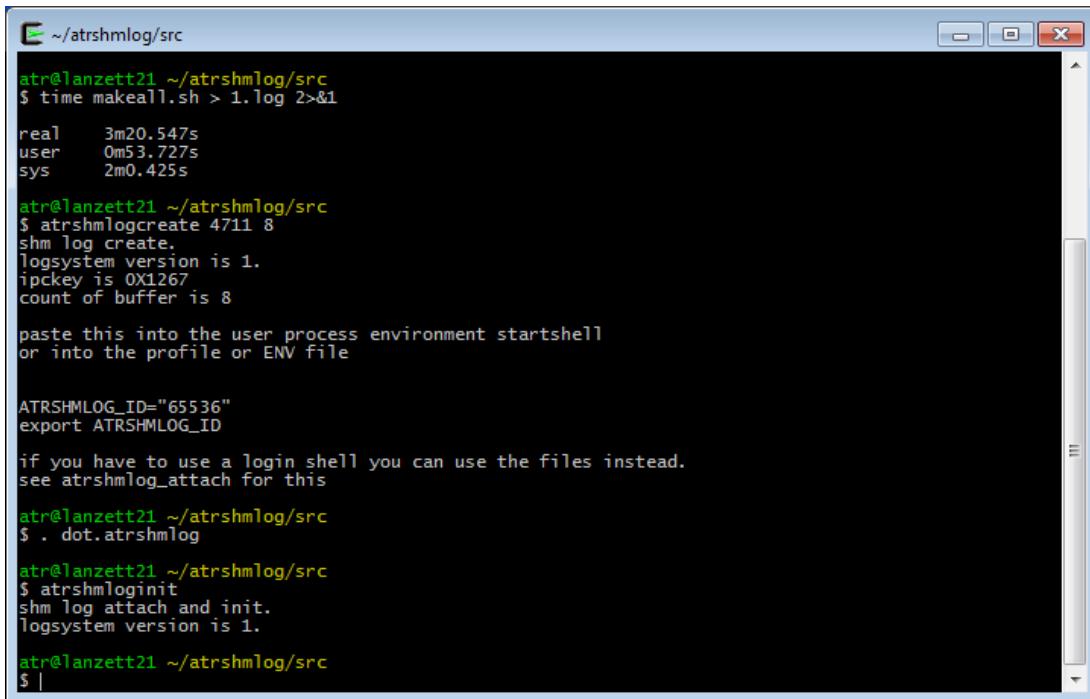
if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
atr@lanzett21 ~/atrshmlog/src
$ . dot.atrshmlog
atr@lanzett21 ~/atrshmlog/src
$
```

*Illustration 76: Create the shared memory buffer*

We can set the environment with the dot file.

## Making the area with init

Now we init as before:



The screenshot shows a terminal window with the following session:

```
~/atrshmlog/src
atr@lanzett21 ~/atrshmlog/src
$ time makeall.sh > 1.log 2>&1
real    3m20.547s
user    0m53.727s
sys     2m0.425s
atr@lanzett21 ~/atrshmlog/src
$ atrshmlogcreate 4711 8
shm log create.
logsystem version is 1.
ipckey is 0X1267
count of buffer is 8
paste this into the user process environment startshell
or into the profile or ENV file

ATRSHMLOG_ID="65536"
export ATRSHMLOG_ID

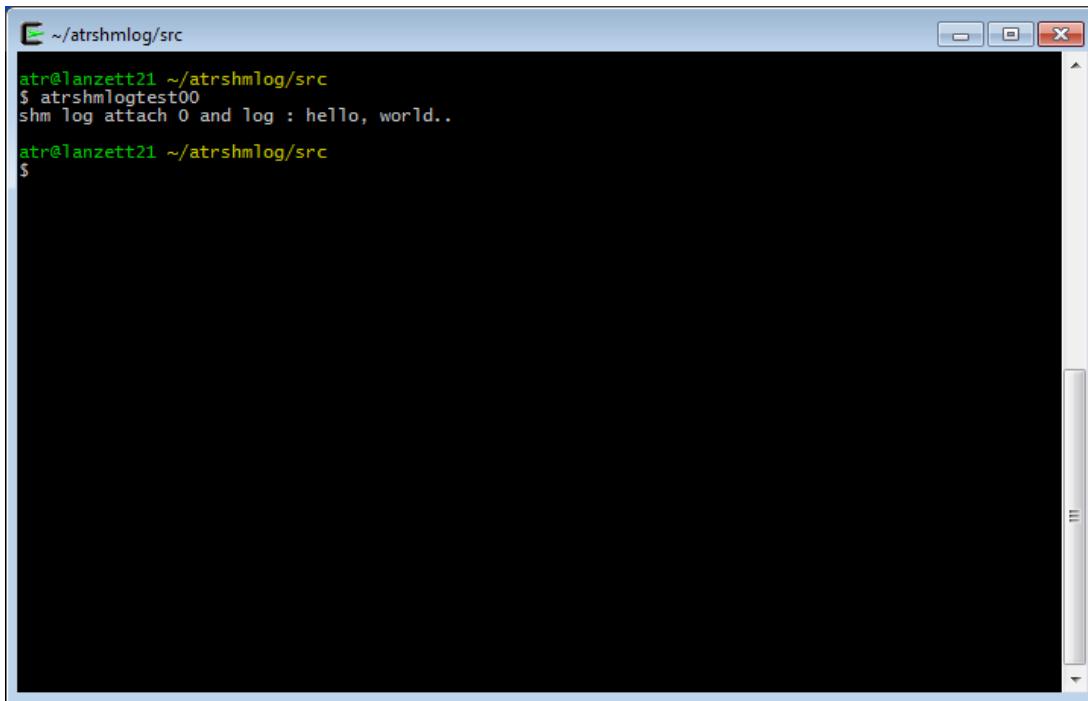
if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this
atr@lanzett21 ~/atrshmlog/src
$ . dot.atrshmlog
atr@lanzett21 ~/atrshmlog/src
$ atrshmloginit
shm log attach and init.
logsystem version is 1.
atr@lanzett21 ~/atrshmlog/src
$ |
```

*Illustration 77: And initialize the area*

Its the same. We use the count already set by the dot file here.

## First test with atrshmlogtest00

And here we go for the first test.

A screenshot of a terminal window titled '~/atrshmlog/src'. The window contains the following text:

```
atr@lanzett21 ~/atrshmlog/src
$ atrshmlogtest00
shm log attach 0 and log : hello, world..
atr@lanzett21 ~/atrshmlog/src
$
```

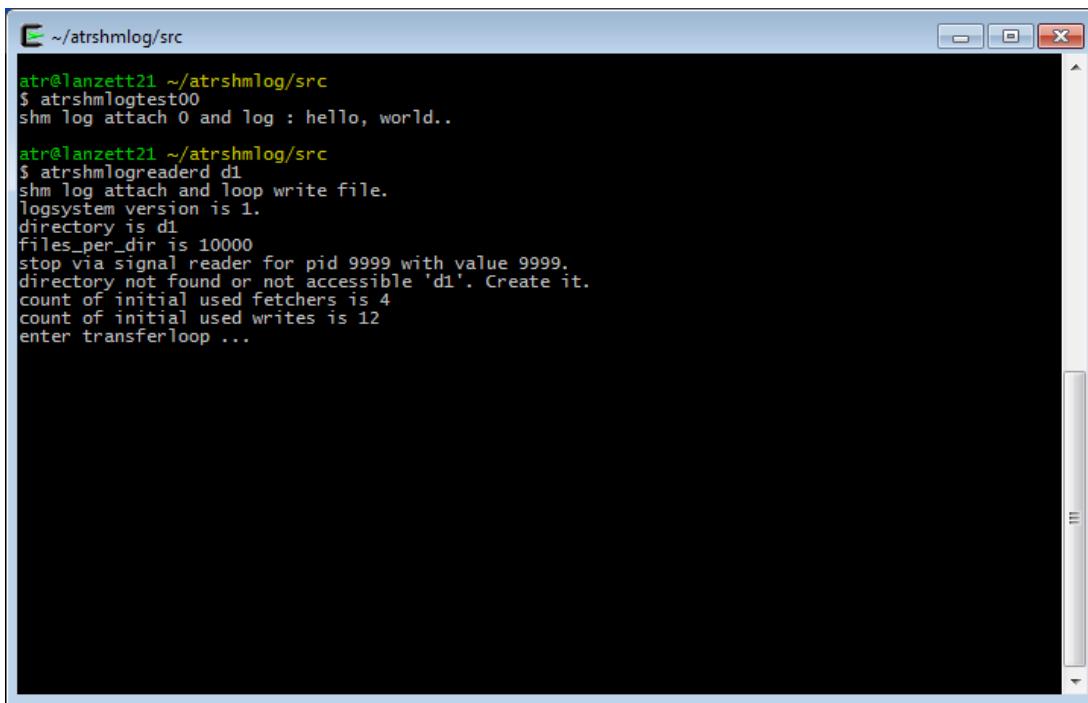
The terminal has a light blue border and a dark background. The scroll bar on the right side shows some minor scroll marks.

*Illustration 78: Running the first test*

So far, so good. Now we have to do the transfer into file system.

## Reader for transfer

That's when the readerd comes in again.



```
~/atrshmlog/src
atr@lanzett21 ~/atrshmlog/src
$ atrshmlogtest00
shm log attach 0 and log : hello, world..

atr@lanzett21 ~/atrshmlog/src
$ atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
enter transferloop ...
```

*Illustration 79: Starting the reader to transfer the log*

So we stop it after the connect is done and the transfer happened.

The second shell with the atrshmlogstopreader.

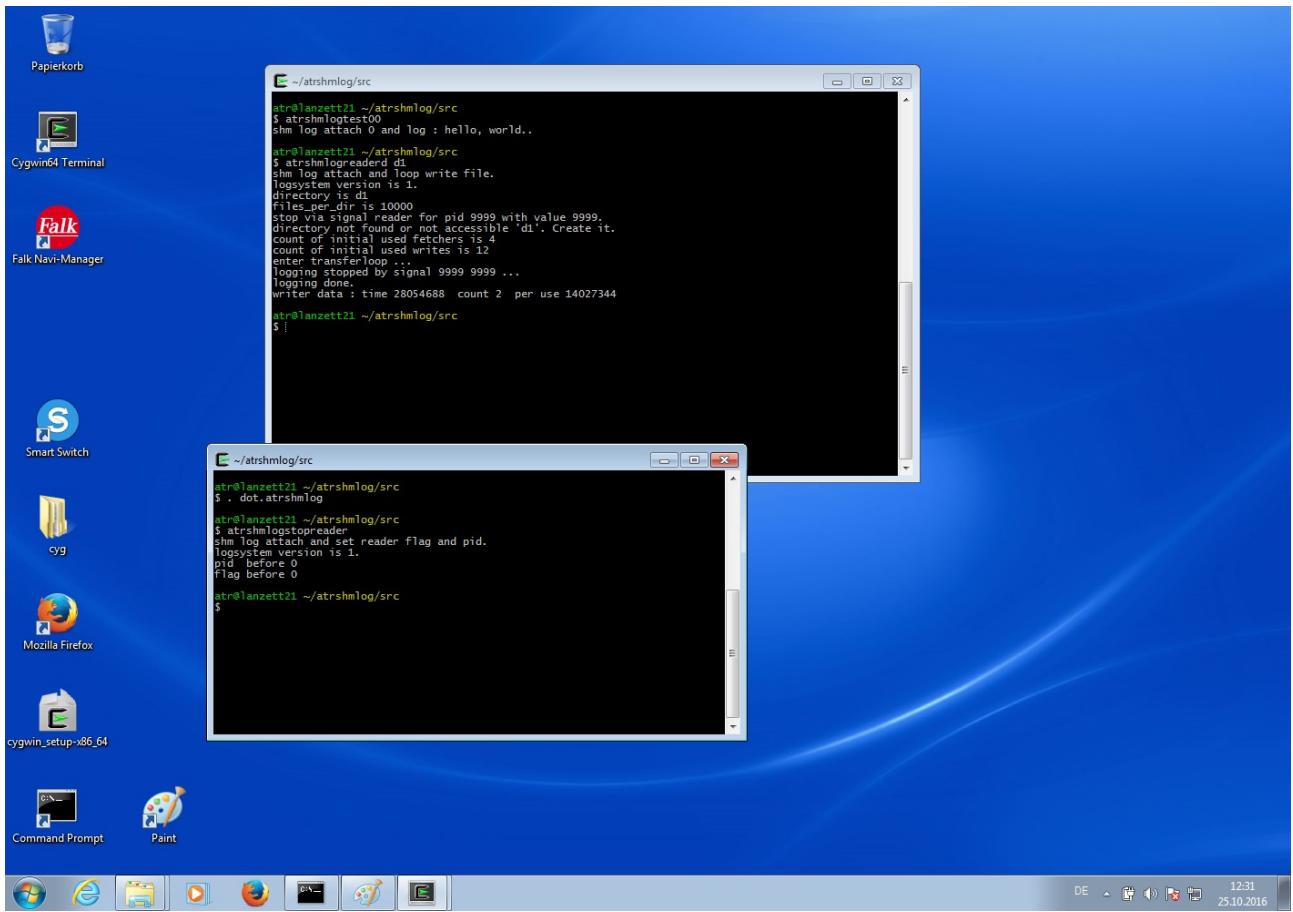
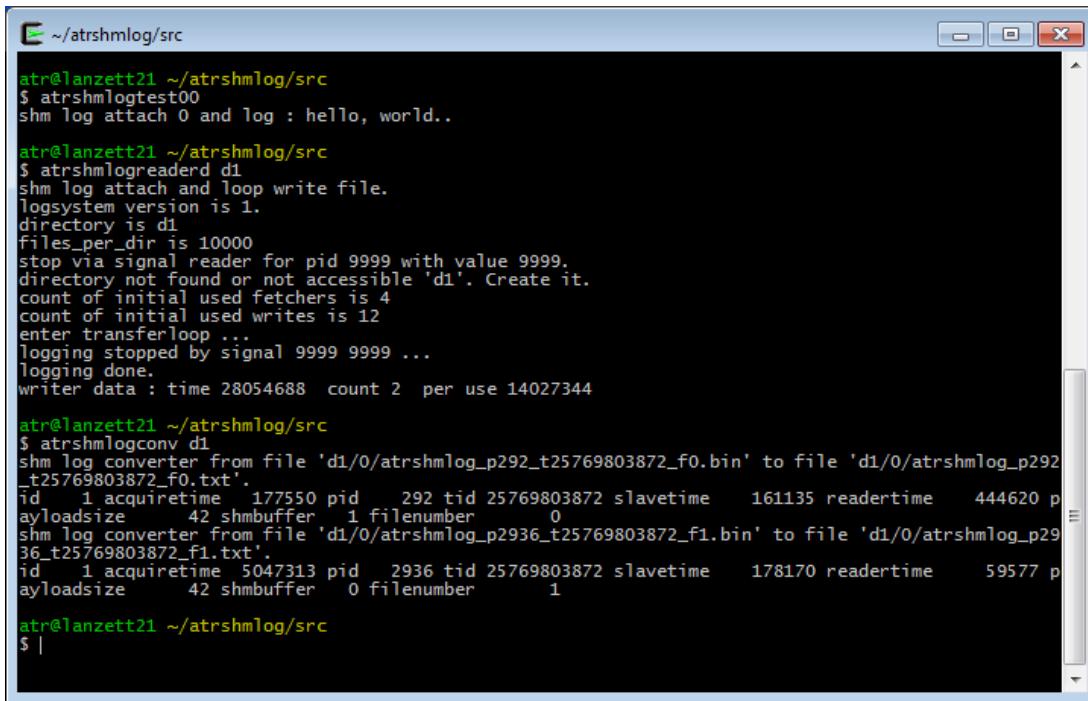


Illustration 80: Stopping the reader after its done.

OK. It stopped. Now we convert the log.

## Conversion of the binary to human readable form

The convert with atrshmlogconv



A screenshot of a terminal window titled '~/atrshmlog/src'. The window contains the following text output:

```
atr@lanzett21 ~/atrshmlog/src
$ atrshmlogtest00
shm log attach 0 and log : hello, world..

atr@lanzett21 ~/atrshmlog/src
$ atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
enter transferloop ...
Logging stopped by signal 9999 9999 ...
logging done.
writer data : time 28054688 count 2 per use 14027344

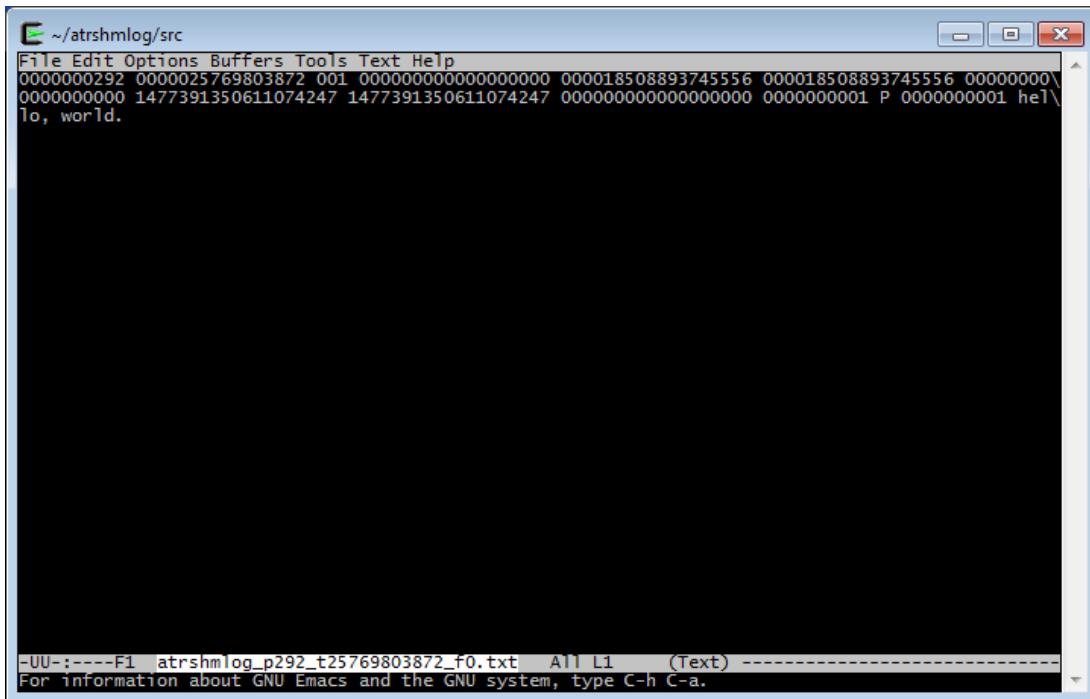
atr@lanzett21 ~/atrshmlog/src
$ atrshmlogconv d1
shm log converter from file 'd1/0/atrshmlog_p292_t25769803872_f0.bin' to file 'd1/0/atrshmlog_p292_t25769803872_f0.txt'.
id 1 acquiretime 177550 pid 292 tid 25769803872 slavetime 161135 readertime 444620 p
ayloadsize 42 shmbuffer 1 filenumber 0
shm log converter from file 'd1/0/atrshmlog_p2936_t25769803872_f1.bin' to file 'd1/0/atrshmlog_p2936_t25769803872_f1.txt'.
id 1 acquiretime 5047313 pid 2936 tid 25769803872 slavetime 178170 readertime 59577 p
ayloadsize 42 shmbuffer 0 filenumber 1

atr@lanzett21 ~/atrshmlog/src
$ |
```

*Illustration 81: Convert from binary to human readable text*

So we have now the conversion done. Last thing is to check the resulting file .

Check the file ....



A screenshot of a terminal window titled '~/atrshmlog/src'. The window contains a large amount of binary or hex dump data at the top, followed by a single line of text: 'Hello, world.' Below the terminal window, the status bar displays the file name 'attrshmlog\_p292\_t25769803872\_f0.txt' and the mode '(Text)'. The status bar also includes standard Emacs keybinding information: F1 for help, C-h for command history, and C-a for beginning of line.

*Illustration 82: The result log*

OK. We have done it for cygwin, too.

## **Another platform : mingw**

For the use of the module we don't need a C11 compiler – simply linking the library is enough. For example we can then use it for ANSI C or even worse thing.

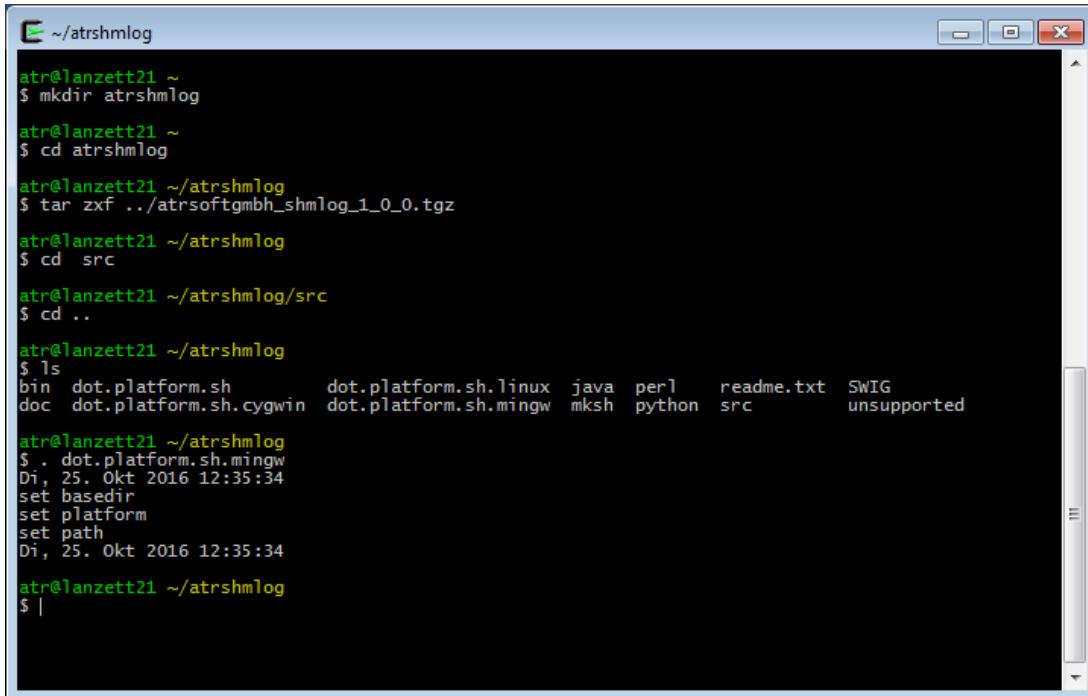
So we can build it with the mingw cross compiler that is part of the cygwin system and use it in the C programs for the fenster;plural platform.

This has a big advantage – we don't have to adapt the scripts to another platform. We can simply use them as in posix or cygwin environment builds. Only the compiler and linker stuff is different. And so there is already a platform dot file for this.

One warning: If you have build a cygwin version you should NOT build in the same place the mingw version. Take another directory and start there. In this example I had killed the cygwin port and started for an empty directory from ground. So don't mix them.

The rest is still the 1.0.0 version – so much for my love for mingw. You simply have to use the cygwin dot file and no copy of headers any more. The rest is simply the same (ok, there is the tests now, but I hope you can figure that out by yourself).

Here is the basedir



The screenshot shows a terminal window titled '~/atrshmlog' running on a Cygwin system. The user has navigated to the directory and extracted a tarball. They then cd'd into the 'src' directory and ran a script named 'dot.platform.sh.mingw'. The output of this script sets environment variables like 'basedir', 'platform', and 'path'. Finally, the user runs a command that outputs '|'. The terminal window has a standard Windows-style title bar and scroll bars.

```
~/atrshmlog
$ mkdir atrshmlog
atr@lanzett21 ~
$ cd atrshmlog
atr@lanzett21 ~/atrshmlog
$ tar zxf ../atrssoftgmbh_shmlog_1_0_0.tgz
atr@lanzett21 ~/atrshmlog
$ cd src
atr@lanzett21 ~/atrshmlog/src
$ cd ..
atr@lanzett21 ~/atrshmlog
$ ls
bin  dot.platform.sh      dot.platform.sh.linux  java  perl  readme.txt  SWIG
doc  dot.platform.sh.cygwin  dot.platform.sh.mingw  mksh  python  src      unsupported

atr@lanzett21 ~/atrshmlog
$ . dot.platform.sh.mingw
Di, 25. Okt 2016 12:35:34
set basedir
set platform
set path
Di, 25. Okt 2016 12:35:34

atr@lanzett21 ~/atrshmlog
$ |
```

*Illustration 83: The mingw base directory in a cygwin system and setting the environment*

The setting of the environment variables done with

dot.platform.sh.mingw

as source file.

## Copy the headers – NO MORE ...

We copy the already adapted headers from the alreadythere directory.

```
~/atrshmlog/src
atr@lanzett21 ~
$ cd atrshmlog/
atr@lanzett21 ~/atrshmlog
$ . dot.platform.sh.mingw
Di, 25. Okt 2016 12:38:33
set basedir
set platform
set path
Di, 25. Okt 2016 12:38:34
atr@lanzett21 ~/atrshmlog
$ cd src
atr@lanzett21 ~/atrshmlog/src
$ cp alreadythere/atrshmlog.h.mingw atrshmlog.h
atr@lanzett21 ~/atrshmlog/src
$ cp alreadythere/atrshmlog_internal.h.mingw atrshmlog_internal.h
atr@lanzett21 ~/atrshmlog/src
$
```

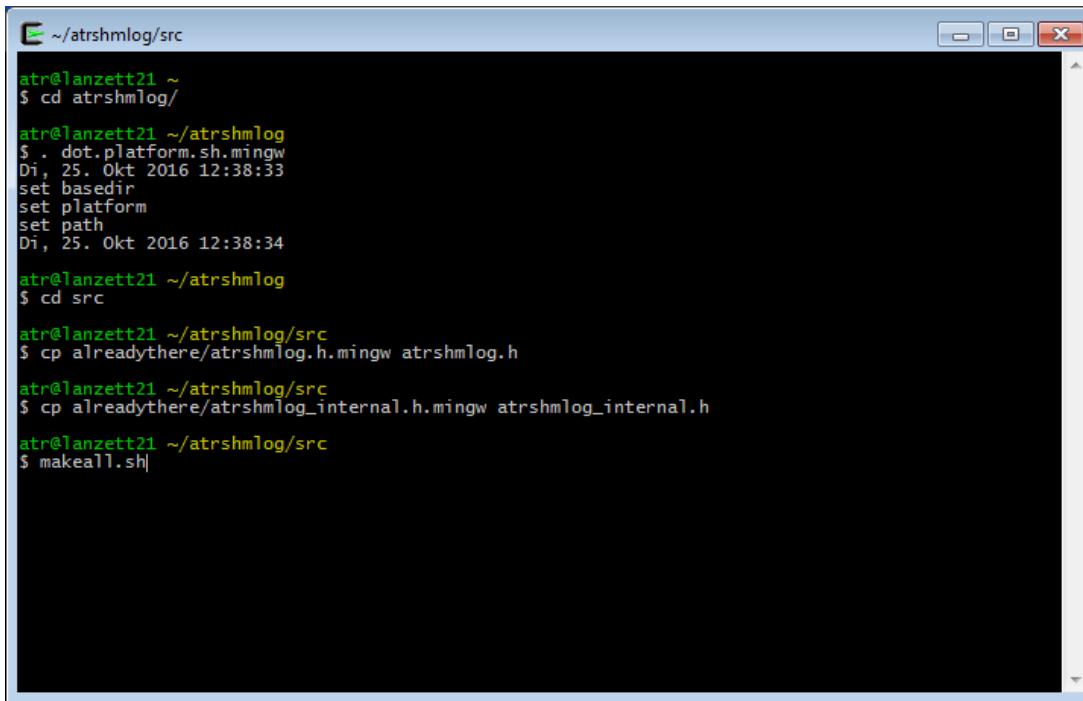
*Illustration 84: Copy of the already adapted headers*

OK. So we are ready for the makeall.sh now.

THATS NO LONGER NEEDED. We do this with the platform now.

## Compile with makeall.sh

Nothing special. Simply as always the makeall.sh.



The screenshot shows a terminal window titled '~/atrshmlog/src' running on a Linux system. The terminal output is as follows:

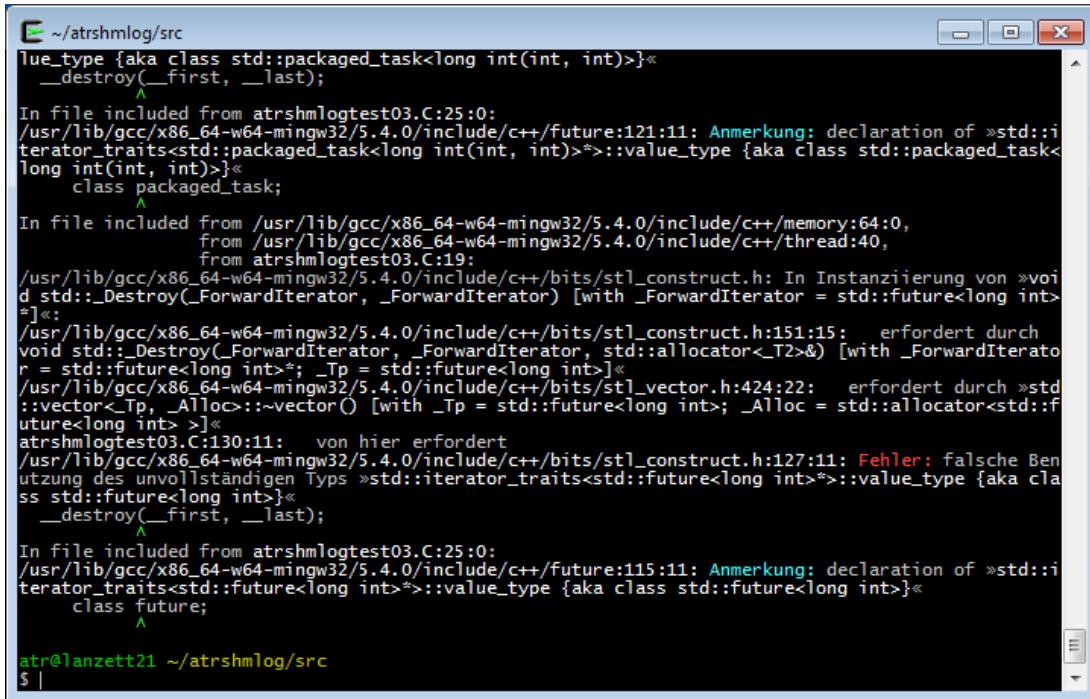
```
atr@lanzett21 ~
$ cd atrshmlog/
atr@lanzett21 ~/atrshmlog
$ . dot.platform.sh.mingw
Di, 25. Okt 2016 12:38:33
set basedir
set platform
set path
Di, 25. Okt 2016 12:38:34
atr@lanzett21 ~/atrshmlog
$ cd src
atr@lanzett21 ~/atrshmlog/src
$ cp alreadythere/atrshmlog.h.mingw atrshmlog.h
atr@lanzett21 ~/atrshmlog/src
$ cp alreadythere/atrshmlog_internal.h.mingw atrshmlog_internal.h
atr@lanzett21 ~/atrshmlog/src
$ makeall.sh|
```

*Illustration 85: Starting the build with makeall.sh*

And this time after 9 minutes.... ( that was before we added the tests and t\_test.sh ...)

## End of compile

We crash land for the last program, its the C++ compiler problem with threads via pthread – we didn't turn that on because we use win threads. If you check the output all is normal till you hit the atrshmlogtest03.C.



The screenshot shows a terminal window with the following text output:

```
~/atrshmlog/src
lue_type {aka class std::packaged_task<long int(int, int)>}<
    _destroy(__first, __last);
In file included from atrshmlogtest03.C:25:0:
/usr/lib/gcc/x86_64-w64-mingw32/5.4.0/include/c++/future:121:11: Anmerkung: declaration of »std::iterator_traits<std::packaged_task<long int(int, int)>*>::value_type {aka class std::packaged_task<long int(int, int)>}<
    class packaged_task;
In file included from /usr/lib/gcc/x86_64-w64-mingw32/5.4.0/include/c++/memory:64:0,
                 from /usr/lib/gcc/x86_64-w64-mingw32/5.4.0/include/c++/thread:40,
                 from atrshmlogtest03.C:19:
/usr/lib/gcc/x86_64-w64-mingw32/5.4.0/include/c++/bits/stl_construct.h: In Instanzierung von »void std::Destroy(_ForwardIterator, _ForwardIterator) [with _ForwardIterator = std::future<long int> *]«:
/usr/lib/gcc/x86_64-w64-mingw32/5.4.0/include/c++/bits/stl_construct.h:151:15:    erfordert durch
void std::Destroy(_ForwardIterator, _ForwardIterator, std::allocator<T2>&) [with _ForwardIterato
r = std::future<long int>*; _Tp = std::future<long int>]<
/usr/lib/gcc/x86_64-w64-mingw32/5.4.0/include/c++/bits/stl_vector.h:424:22:    erfordert durch »std
::vector<_Tp, _Alloc>::~vector() [with _Tp = std::future<long int>; _Alloc = std::allocator<std::f
uture<long int> >]<
atrshmlogtest03.C:130:11:    von hier erfordert
/usr/lib/gcc/x86_64-w64-mingw32/5.4.0/include/c++/bits/stl_construct.h:127:11: Fehler: falsche Ben
utzung des unvollständigen Typs »std::iterator_traits<std::future<long int>*>::value_type {aka cla
ss std::future<long int>}<
    _destroy(__first, __last);
In file included from atrshmlogtest03.C:25:0:
/usr/lib/gcc/x86_64-w64-mingw32/5.4.0/include/c++/future:115:11: Anmerkung: declaration of »std::i
terator_traits<std::future<long int>*>::value_type {aka class std::future<long int>}<
    class future;
attr@lanzett21 ~/atrshmlog/src
$ |
```

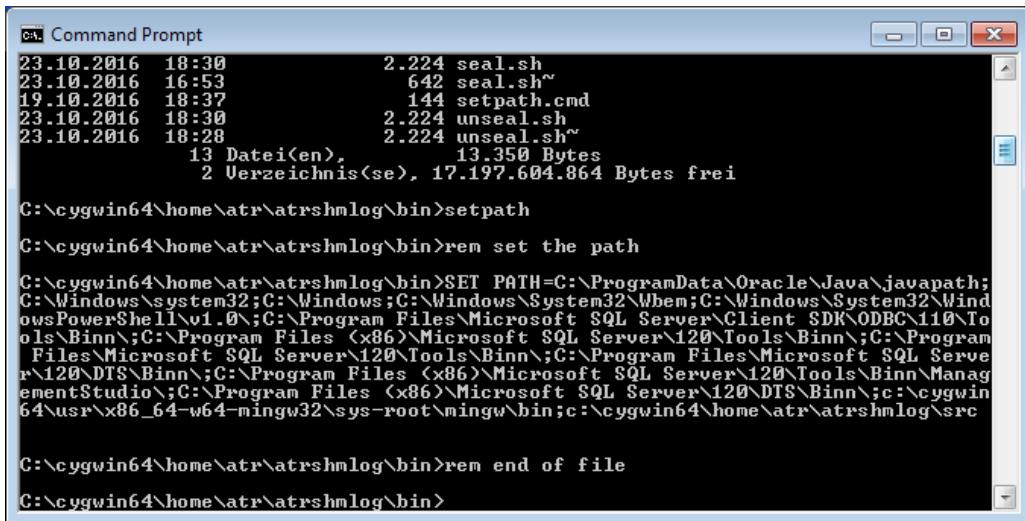
Illustration 86: A crash landing for test03 ....

There is a patched shmCPPfiles list file in case you don't like a crash landing at the end. You can copy it together with the includes and no 03 program will be build then .

Sad, but not a real problem. The rest is done. So we can start with testing.

## Path handling for vanilla cmd

We use the programs directly from a fresh vanilla cmd – no shell this time. If you insist you can do this on another box where no cygwin is installed ( not quite right – you will still need the compiler stub dll for mingw, but that's a small price to pay).



```
23.10.2016  18:30          2.224 seal.sh
23.10.2016  16:53          642 seal.sh~
23.10.2016  18:37          144 setpath.cmd
23.10.2016  18:30          2.224 unseal.sh
23.10.2016  18:28          2.224 unseal.sh~
13 Datei(en),           13.350 Bytes
2 Verzeichnis(se), 17.197.604.864 Bytes frei

C:\cygwin64\home\atr\atrhsmlog\bin>setpath
C:\cygwin64\home\atr\atrhsmlog\bin>rem set the path
C:\cygwin64\home\atr\atrhsmlog\bin>SET PATH=C:\ProgramData\Oracle\Java\javapath;
C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\110\Tools\Binn\;C:\Program Files (x86)\Microsoft SQL Server\120\Tools\Binn\;C:\Program Files\Microsoft SQL Server\120\Tools\Binn\;C:\Program Files\Microsoft SQL Server\120\DTSP\Binn\;C:\Program Files (x86)\Microsoft SQL Server\120\Tools\Binn\ManagementStudio\;C:\Program Files (x86)\Microsoft SQL Server\120\DTSP\Binn\;c:\cygwin64\usr\x86_64-w64-mingw32\sys-root\mingw\bin;c:\cygwin64\home\atr\atrhsmlog\src

C:\cygwin64\home\atr\atrhsmlog\bin>rem end of file
C:\cygwin64\home\atr\atrhsmlog\bin>
```

*Illustration 87: Setting the path for a cmd*

OK. Setting a path is after all not so dramatic. Simply check for your installation if the path is not the same. Now its time to start something.

## Creating the buffer

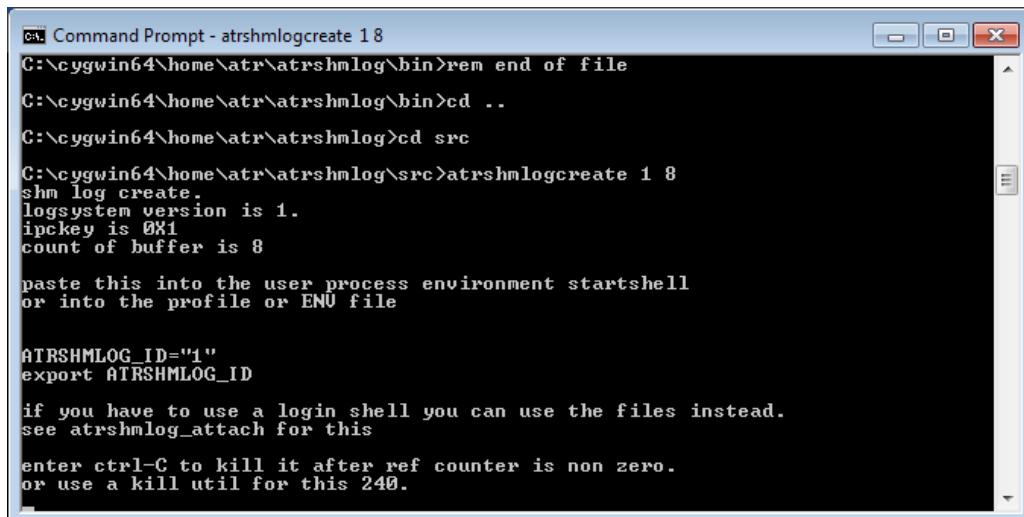
We create the buffer as always with atrshmlogcreate. But this time its different for the parameters. We need here an internal info. The shared memory approach of fenster;plural is the same as a memory mapped file. So we need a file name for the thing and in theory a real file in the file system.

To make things more difficult there are so called Global and Local files. The Global files can be used from anybody, the Local only from sessions for the same User.

So if you try to understand the cygwin thing – bingo. A global file approach to cover the shared memory.

So I implemented 16 Local and Global file names in the module, and you get them via index. To make the 0 not a problem its indexed from 1 to 32. And the attach then uses that index via parameter - and if you are lucky you simply get the parameter back for the user programs.

So our first parameter is 1. The second is still the number of buffers. I use the 8 again.



```
ca. Command Prompt - atrshmlogcreate 1 8
C:\cygwin64\home\atr\atrshmlog\bin>rem end of file
C:\cygwin64\home\atr\atrshmlog\bin>cd ..
C:\cygwin64\home\atr\atrshmlog>cd src
C:\cygwin64\home\atr\atrshmlog\src>atrshmlogcreate 1 8
shm log create.
logsystem version is 1.
ipckey is 0x1
count of buffer is 8

paste this into the user process environment startshell
or into the profile or ENU file

ATRSHMLOG_ID="1"
export ATRSHMLOG_ID

if you have to use a login shell you can use the files instead.
see atrshmlog_attach for this

enter ctrl-C to kill it after ref counter is non zero.
or use a kill util for this 240.
```

Illustration 88: Create of a shared mapped memory via pagefile.sys

Create done, but....

.... the program hangs. It will not end. And that has to do with the fact that we didn't used any real file here. So the mapping was done in fact for part of the page file. That's a bid wired, but it works for me, so I didn't made the other thing.

OK. So why it hangs ?

Well. The mapping for the page file is as all mappings done with reference counting. So if there is no process holding a reference – the kernel will simply delete the mapping and set free the memory.

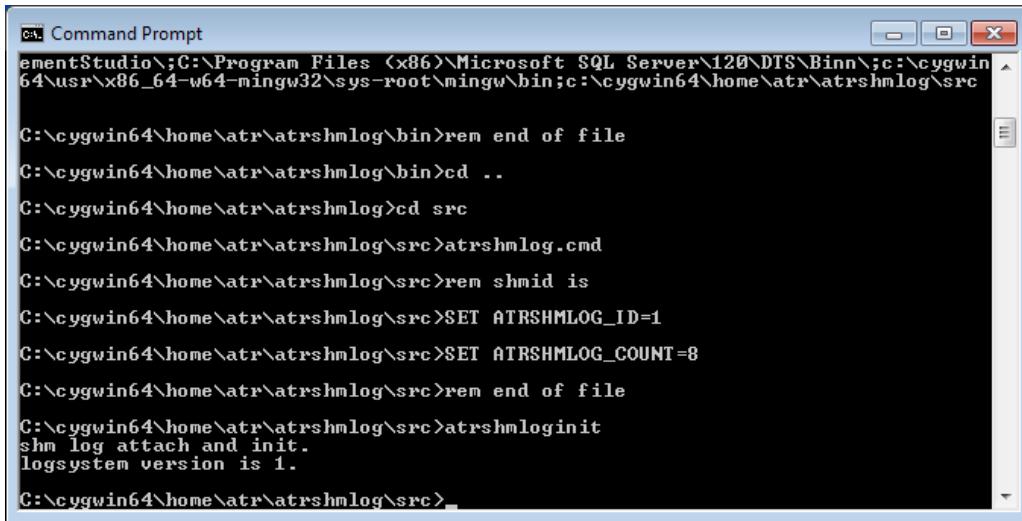
So if we don't hang – we would likely end and the mapped memory freed before we had a chance to connect with other programs.

Bingo. That's why the cygwin uses a service for the cygrunsrv – its a permanent running program in the background, so it can map and then still hold the mapped file till someone is interested in it.

So in the end – without a backup file in the file system its simply the easiest thing to let the program not die after the create till at least another one has connected.

## The init of the area

We now can use another cmd and init then the area as usual. We use the dot file – this time atrshmlog.cmd – to set the variables for the thing.



```
C:\ Command Prompt
e:mentStudio\;C:\Program Files (x86)\Microsoft SQL Server\120\DT
S\Binn\;c:\cygwin64\usr\x86_64-w64-mingw32\sys-root\mingw\bin;c:\cygwin64\home\atr\atrshmlog\src

C:\cygwin64\home\atr\atrshmlog\bin>rem end of file
C:\cygwin64\home\atr\atrshmlog\bin>cd ..
C:\cygwin64\home\atr\atrshmlog>cd src
C:\cygwin64\home\atr\atrshmlog\src>atrshmlog.cmd
C:\cygwin64\home\atr\atrshmlog\src>rem shmid is
C:\cygwin64\home\atr\atrshmlog\src>SET ATRSHMLOG_ID=1
C:\cygwin64\home\atr\atrshmlog\src>SET ATRSHMLOG_COUNT=8
C:\cygwin64\home\atr\atrshmlog\src>rem end of file
C:\cygwin64\home\atr\atrshmlog\src>atrshmloginit
shm log attach and init.
logsystem version is 1.

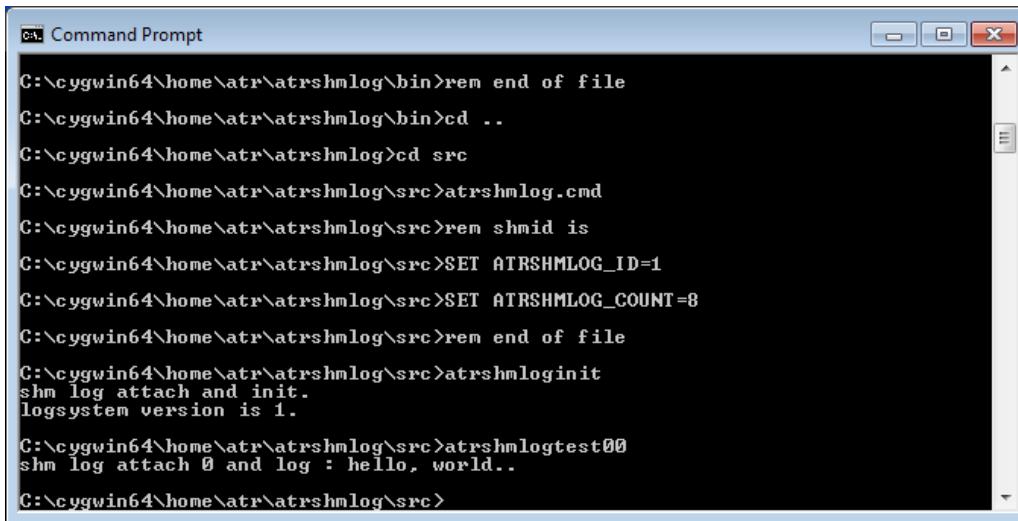
C:\cygwin64\home\atr\atrshmlog\src>
```

*Illustration 89: Initialize or the area from a cmd*

OK. This is more or less the usual now. Important is that we NOT kill our create process in this version of the game before we run the other programs.

## First test with atrshmlogtest00

Its time now for our first logging test.



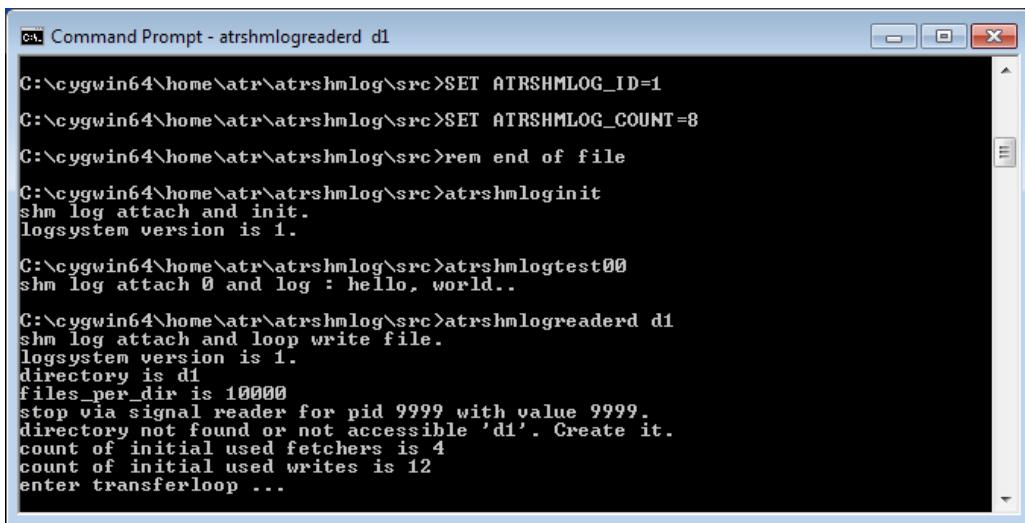
```
C:\cygwin64\home\atr\atrshmlog\bin>rem end of file
C:\cygwin64\home\atr\atrshmlog\bin>cd ..
C:\cygwin64\home\atr\atrshmlog>cd src
C:\cygwin64\home\atr\atrshmlog\src>atrshmlog.cmd
C:\cygwin64\home\atr\atrshmlog\src>rem shmid is
C:\cygwin64\home\atr\atrshmlog\src>SET ATRSHMLOG_ID=1
C:\cygwin64\home\atr\atrshmlog\src>SET ATRSHMLOG_COUNT=8
C:\cygwin64\home\atr\atrshmlog\src>rem end of file
C:\cygwin64\home\atr\atrshmlog\src>atrshmloginit
shm log attach and init.
logsystem version is 1.
C:\cygwin64\home\atr\atrshmlog\src>atrshmlogtest00
shm log attach 0 and log : hello, world..
C:\cygwin64\home\atr\atrshmlog\src>
```

*Illustration 90: Running a first test from a cmd*

OK. This went as always, the program did at least tell us that. So we get the memory next.

## Starting the reader for memory fetching

We prepare a new cmd and start the reader.



```
ca. Command Prompt - atrshmlogreaderd d1
C:\cygwin64\home\atr\atrshmlog\src>SET ATRSHMLOG_ID=1
C:\cygwin64\home\atr\atrshmlog\src>SET ATRSHMLOG_COUNT=8
C:\cygwin64\home\atr\atrshmlog\src>rem end of file
C:\cygwin64\home\atr\atrshmlog\src>atrshmloginit
shm log attach and init.
logsystem version is 1.
C:\cygwin64\home\atr\atrshmlog\src>atrshmlogtest00
shm log attach 0 and log : hello, world..
C:\cygwin64\home\atr\atrshmlog\src>atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
enter transferloop ...
```

*Illustration 91: Running the reader from a cmd*

OK. Nothing dramatic different so far. In theory we could now get rid of the create – the reader is connected – but to make things not to complicated we leave the create where it is.

For the data we have now to stop the reader and then check for the file.

Stopping the reader with atrshmlogsignalreader.

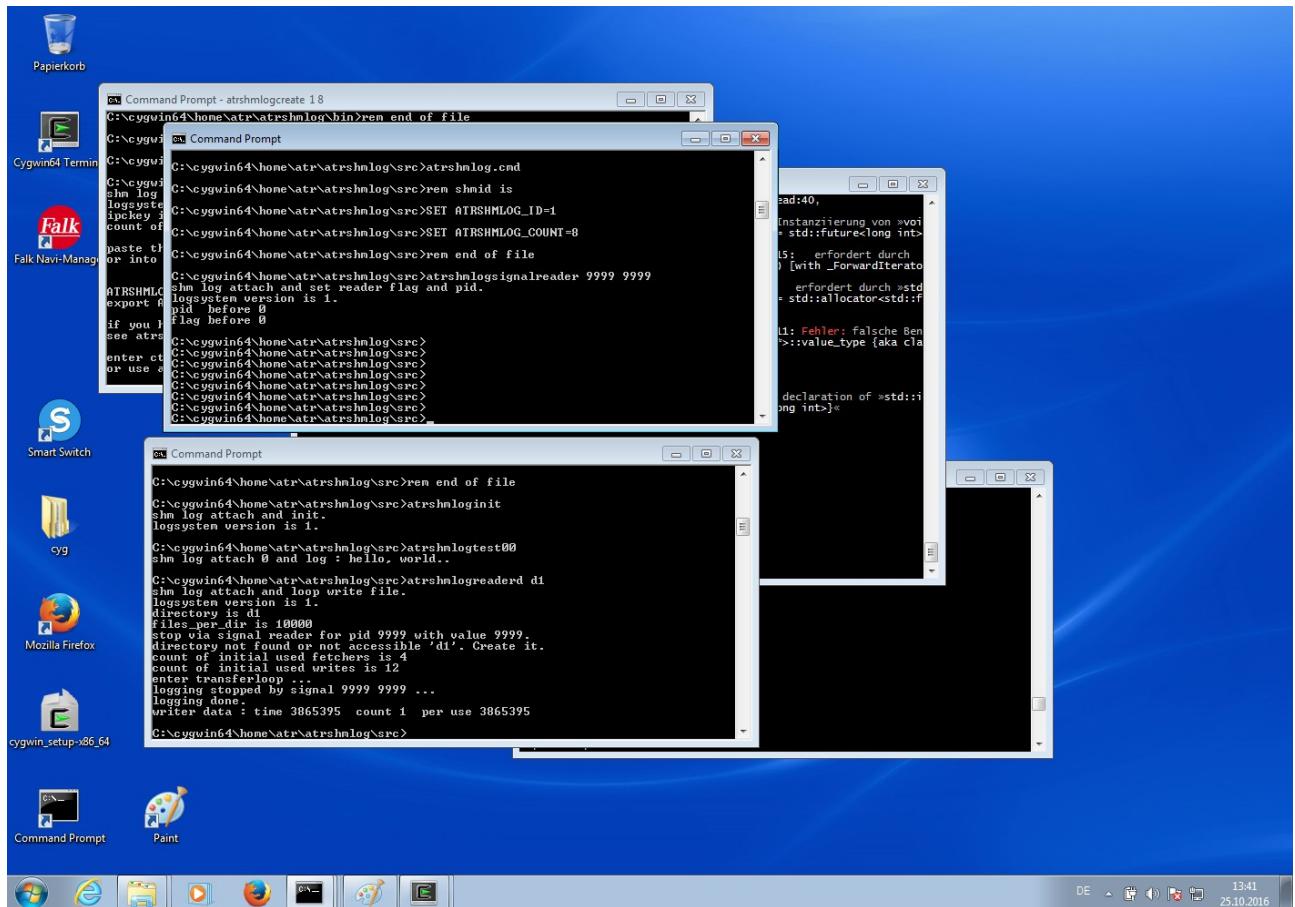


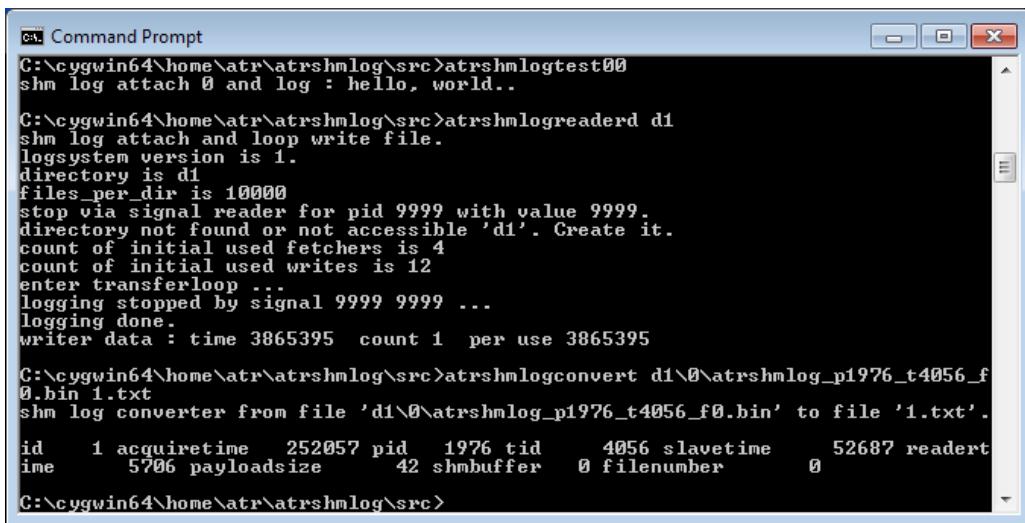
Illustration 92: Using signal reader to stop the reader - from a cmd

This worked. But I don't have made a cmd for it, so we had to use the raw signal reader instead of the atrshmlogstopreader script as it is for the other platform.

Try to adapt it and see for yourself how its done.

## Conversion of the binary into human readable text

Again I don't have a cmd in place, so we have to use a raw program call here.



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The command entered is "atrshmlogconvert d1\0\atrshmlog\_p1976\_t4056\_f0.bin 1.txt". The output displays the conversion process, including the creation of the file "1.txt" from the binary log file "d1\0\atrshmlog\_p1976\_t4056\_f0.bin". The output also includes some internal log information and statistics about the log entries.

```
C:\cygwin64\home\atr\atrshmlog\src>atrshmlogtest00
shm log attach 0 and log : hello, world..

C:\cygwin64\home\atr\atrshmlog\src>atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 4
count of initial used writes is 12
enter transferloop ...
logging stopped by signal 9999 9999 ...
logging done.
writer data : time 3865395 count 1 per use 3865395

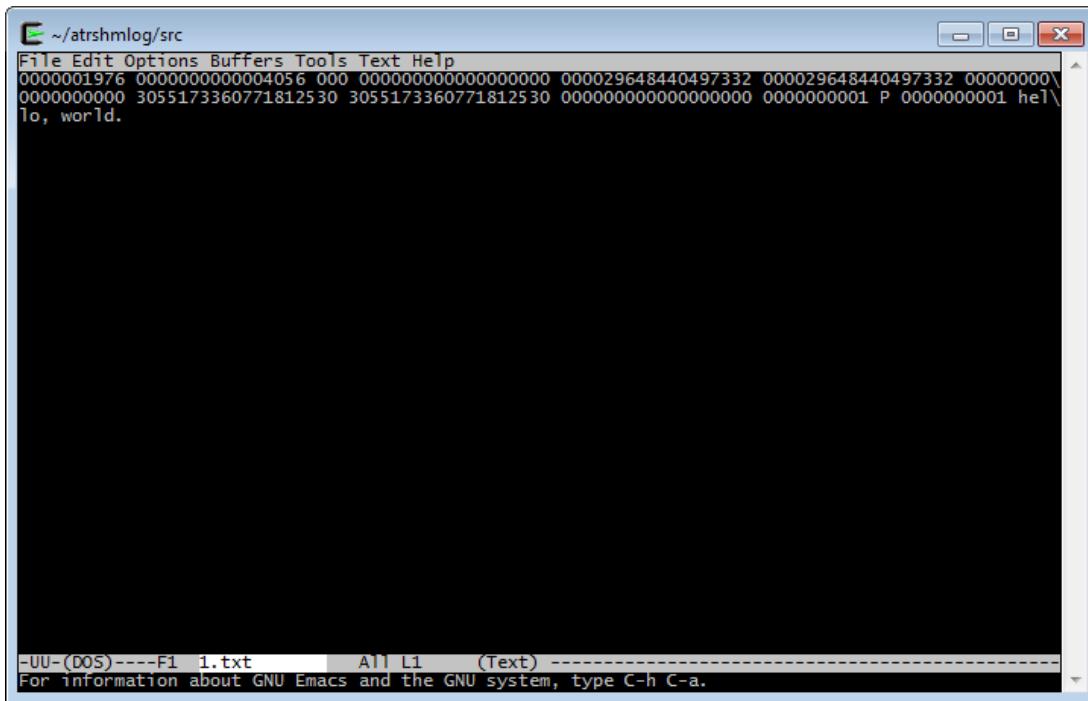
C:\cygwin64\home\atr\atrshmlog\src>atrshmlogconvert d1\0\atrshmlog_p1976_t4056_f0.bin 1.txt
shm log converter from file 'd1\0\atrshmlog_p1976_t4056_f0.bin' to file '1.txt'.
id    1 acquiretime 252057 pid   1976 tid     4056 slavetime      52687 readert
ime   5706 payloadsize    42 shmbuffer    0 filenumber      0

C:\cygwin64\home\atr\atrshmlog\src>
```

*Illustration 93: Using the convert from a cmd to get human readable text*

This time its in the directory src and its name is 1.txt. We check now.

Check with my favorite text editor. You can use of course yours if you want.



*Illustration 94: The result log*

OK. We could make the mingw platform so far. Now its time to add something vital.

The java layer.

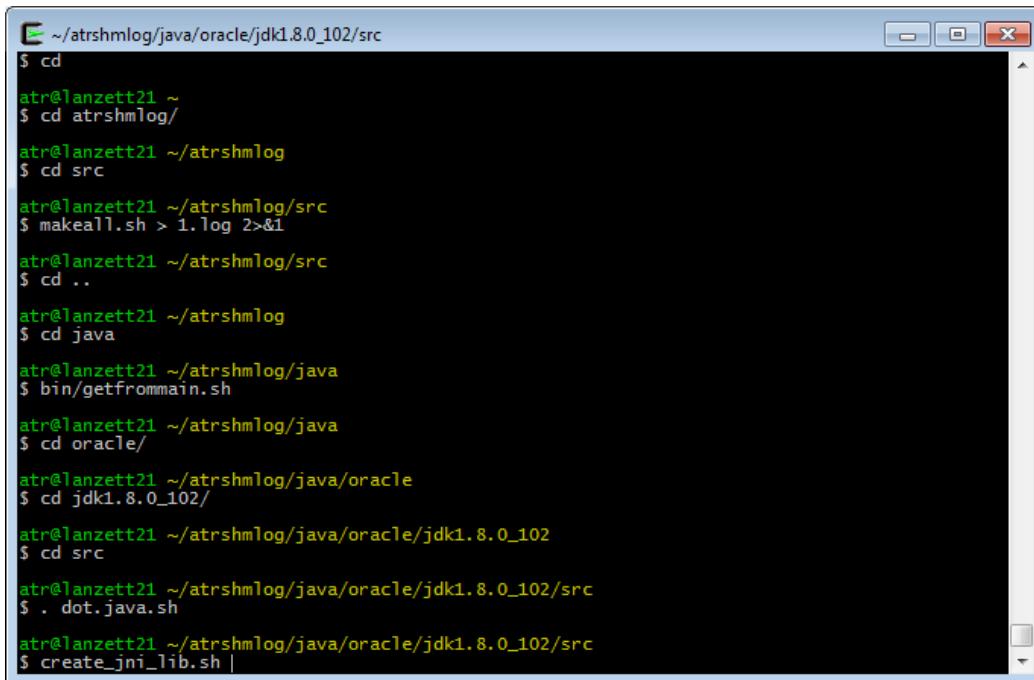
This was also made with the cygwin, but the system didn't made it after starting the jvm for execution. It hang endless. So don't try the jni bridge with cygwin, its no use. Switch to mingw for that.

## The jni layer for mingw

As with the other systems, you do this step by step as for the Linux. Change to the BASEDIR/java and get the library and headers in place with the getfrommain.sh.

Next go to the right vendor and jdk version, enter the src, source the dot.java.sh and then build it with the create\_jni\_lib.sh.

This is done on cygwin environment, but its for the mingw version, don't forget !

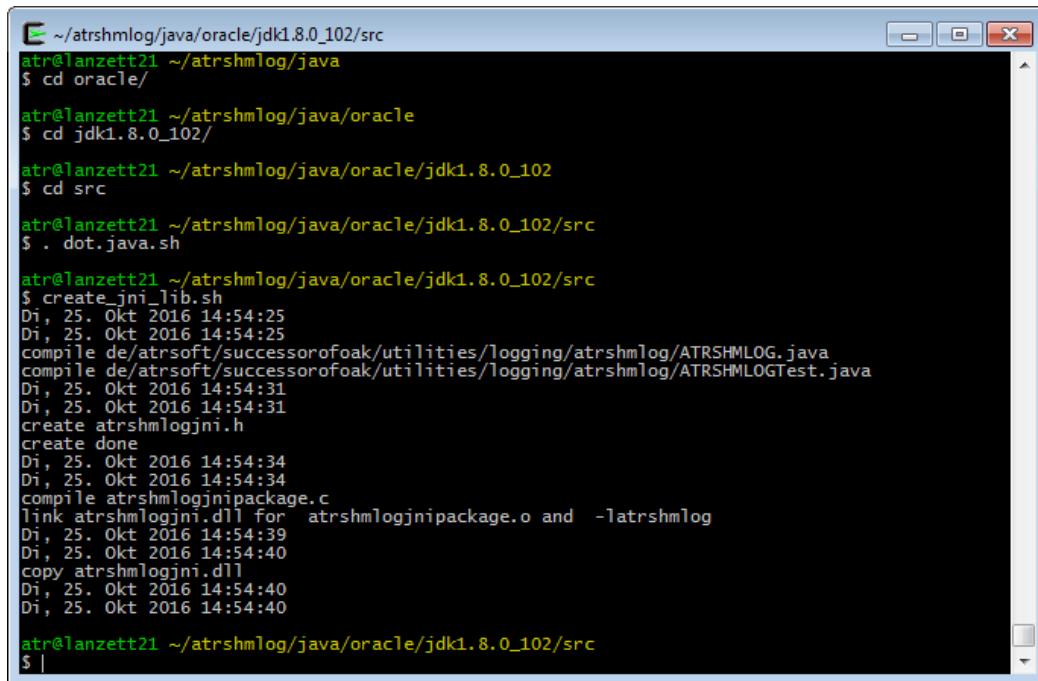


```
~/atrshmlog/java/oracle/jdk1.8.0_102/src
$ cd
atr@lanzett21 ~
$ cd atrshmlog/
atr@lanzett21 ~/atrshmlog
$ cd src
atr@lanzett21 ~/atrshmlog/src
$ makeall.sh > 1.log 2>&1
atr@lanzett21 ~/atrshmlog/src
$ cd ..
atr@lanzett21 ~/atrshmlog
$ cd java
atr@lanzett21 ~/atrshmlog/java
$ bin/getfrommain.sh
atr@lanzett21 ~/atrshmlog/java
$ cd oracle/
atr@lanzett21 ~/atrshmlog/java/oracle
$ cd jdk1.8.0_102/
atr@lanzett21 ~/atrshmlog/java/oracle/jdk1.8.0_102
$ cd src
atr@lanzett21 ~/atrshmlog/java/oracle/jdk1.8.0_102/src
$ . dot.java.sh
atr@lanzett21 ~/atrshmlog/java/oracle/jdk1.8.0_102/src
$ create_jni_lib.sh |
```

*Illustration 95: Generation of the jni bridge with mingw for vanilla java*

OK. We have the oracle and jdk1.8.0\_102 in place now. So the build begins.

Building the class files, then the header and at last the new jni bridge dll.



The screenshot shows a terminal window with a blue title bar containing the path: ~/atrshmlog/java/oracle/jdk1.8.0\_102/src. The window is titled 'atr@lanzett21 ~/atrshmlog/java'. The terminal output is as follows:

```
~/atrshmlog/java/oracle/jdk1.8.0_102/src
atr@lanzett21 ~/atrshmlog/java
$ cd oracle/
atr@lanzett21 ~/atrshmlog/java/oracle
$ cd jdk1.8.0_102/
atr@lanzett21 ~/atrshmlog/java/oracle/jdk1.8.0_102
$ cd src
atr@lanzett21 ~/atrshmlog/java/oracle/jdk1.8.0_102/src
$ . dot.java.sh

atr@lanzett21 ~/atrshmlog/java/oracle/jdk1.8.0_102/src
$ create_jni_lib.sh
Di, 25. Okt 2016 14:54:25
Di, 25. Okt 2016 14:54:25
compile de/atrsoft/runnerofoak/utilities/logging/atrshmlog/ATRSHMLOG.java
compile de/atrsoft/runnerofoak/utilities/logging/atrshmlog/ATRSHMLOGTest.java
Di, 25. Okt 2016 14:54:31
Di, 25. Okt 2016 14:54:31
create atrshmlogjni.h
create done
Di, 25. Okt 2016 14:54:34
Di, 25. Okt 2016 14:54:34
compile atrshmlogjnipackage.c
link atrshmlogjni.dll for atrshmlogjnipackage.o and -latrshmlog
Di, 25. Okt 2016 14:54:39
Di, 25. Okt 2016 14:54:40
copy atrshmlogjni.dll
Di, 25. Okt 2016 14:54:40
Di, 25. Okt 2016 14:54:40

atr@lanzett21 ~/atrshmlog/java/oracle/jdk1.8.0_102/src
$ |
```

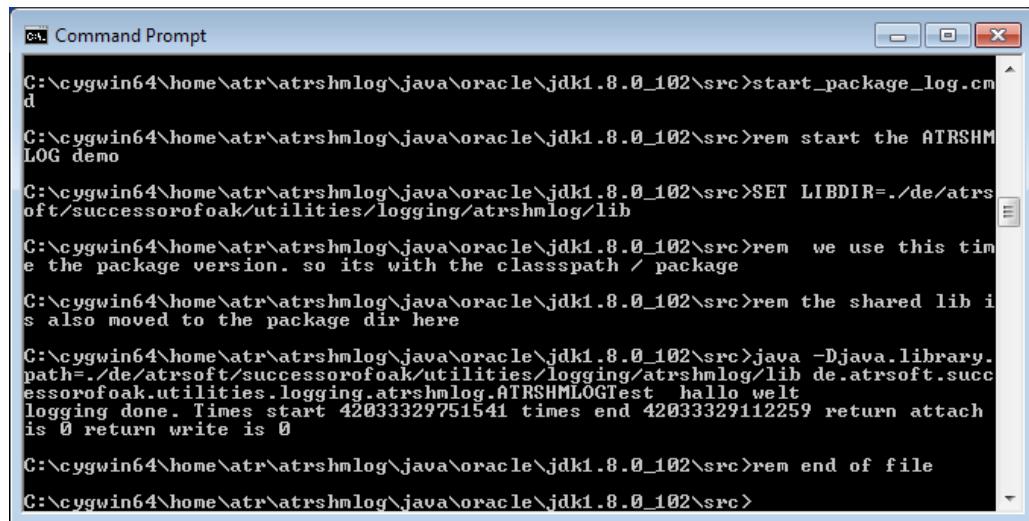
*Illustration 96: Build of the jni bridge for mingw*

OK. We have no abnormality here. Simply as the rest. Its only 15 seconds this time.

And you can see we have the atrshmlogjni.dll this time in the resulting copy.

## Testing the jni bridge for mingw

So now its time for a cmd with both path settings, one for the mingw bin path and one for the load library. We have this in the start script, its this time a cmd.



```
C:\cygwin64\home\atr\atrshmlog\java\oracle\jdk1.8.0_102\src>start_package_log.cmd
C:\cygwin64\home\atr\atrshmlog\java\oracle\jdk1.8.0_102\src>rem start the ATRSHM
LOG demo
C:\cygwin64\home\atr\atrshmlog\java\oracle\jdk1.8.0_102\src>SET LIBDIR=../de/atrso
ft/SuccessorOfOak/utilities/logging/atrshmlog/lib
C:\cygwin64\home\atr\atrshmlog\java\oracle\jdk1.8.0_102\src>rem we use this tim
e the package version. so its with the classspath / package
C:\cygwin64\home\atr\atrshmlog\java\oracle\jdk1.8.0_102\src>rem the shared lib i
s also moved to the package dir here
C:\cygwin64\home\atr\atrshmlog\java\oracle\jdk1.8.0_102\src>java -Djava.library.
path=../de/atrsoft/SuccessorOfOak/utilities/logging/atrshmlog/lib de.atrsoft.succ
essorofOak.utilities.logging.atrshmlog.ATRSHMLOGTest hallo_welt
logging done. Times start 42033329751541 times end 42033329112259 return attach
is 0 return write is 0
C:\cygwin64\home\atr\atrshmlog\java\oracle\jdk1.8.0_102\src>rem end of file
C:\cygwin64\home\atr\atrshmlog\java\oracle\jdk1.8.0_102\src>
```

*Illustration 97: Test of the jni bridge with vanilla java and cmd*

OK. Seems it worked after all. No special things here.

As always, we check the converted result file. I leave the reader and convert stuff this time to the reader ( gag ! ) .

```

~/atrhslog/src
File Edit Options Buffers Tools Text Help
000003632 0000000000003068 000 0000000000000000 00042033329106465 00042033329112259 00000000000005794 3055174392350099277 3055174392350101408 000000\
00000002131 00000000001 I 00000000001
00000003632 0000000000003068 000 0000000000000000 00042033329751541 00042033329757832 00000000000006291 3055174392350336498 3055174392350338812 000000\
0000000214 00000000002 I 00000000001
00000003632 0000000000003068 000 0000000000000000 0004203335065549 00042033340430585 000000000005365036 3055174392352290681 3055174392354263629 000000\
0000012948 0000000003 0000000001 loop 1000 times gettimeofday
00000003632 0000000000003068 000 0000000000000000 00042033340555544 00042033340561755 00000000000006211 3055174392354309582 3055174392354311866 000000\
00000002284 000000000004 0000000000000000 00042033340561755 00042033340568174 00000000000006419 3055174392354311866 3055174392354314227 000000\
00000002361 0000000000000001 0000000000000001
00000003632 0000000000003068 000 0000000000000000 00042033340568174 00042033340573850 00000000000005676 3055174392354314227 3055174392354316314 000000\
0000002087 0000000006 0000000001 Hello, world.
00000003632 0000000000003068 000 0000000000000000 00042033340573850 00042033340587823 00000000000013973 3055174392354316314 3055174392354321452 000000\
00000005138 0000000006 0000000002 time For payload 2 logging
00000003632 0000000000003068 000 0000000000000000 00042033340573850 00042033340593678 00000000000019828 3055174392354316314 3055174392354323605 000000\
00000007291 0000000007 I 0000000001 ADGGJ R-GO
00000003632 0000000000003068 000 0000000000000000 0004203334061408 00042033340675798 000000000000074390 3055174392354326448 3055174392354353804 000000\
000000072356 0000000008 I 0000000001 FAGGJ R-GO
00000003632 0000000000003068 000 0000000000000000 00042033340675798 00042033340683677 00000000000007879 3055174392354353804 3055174392354356702 000000\
00000002598 0000000009 i 0000000001 Hello
00000003632 0000000000003068 000 0000000000000000 00042033340683677 00042033340764182 000000000000080505 3055174392354356702 3055174392354386307 000000\
0000000729601 0000000010 i 0000000001 Hello
00000003632 0000000000003068 000 0000000000000000 00042033340561755 00042033340568174 00000000000006419 3055174392354311866 3055174392354314227 000000\
00000002361 0000000011 I 00000001002
00000003632 0000000000003068 000 0000000000000000 00042033340820574 00042033340825397 00000000000004823 3055174392354407045 3055174392354408818 000000\
000000001773 0000000012 I 0000000001

```

*Illustration 98: The resulting log*

OK. We have a logging jni implementation for mingw, and there is only the stuff in mingw bin that we need beside the created dll so far. I can live with that. If you need it different please contact me and I will see what I can do for you. But for now that's the thing I have for you.

# **What are those numbers for ? Adjustment process ?**

The vanilla module should do the job at least if you are in a simple program scenario. If you have higher needs for performance, or you are simply in a high amount of threads or programs situation the things have to be adjusted to that.

This is called the adjustment process.

So here we try to find out if we can do make things better in this case.

First we see how to find out if there are any needs. Then we check for our options.

## **Now we need to know how fast it is**

The module itself is a simple logging approach when it comes to measure the speed. Simply put an gettimeofday in place, and then print out the clicks. Better the difference.

This way I found that the static (!) allocated memory buffers for the log were slow on fenster;plural if they were not accessed before – a simple log made 3000 to 5000 clicks. When I was inside the first memcpy got it and the second came with 10 clicks ...

So what can we get ?

First of all you have some numbers when you get the timings right from the atrshmlogdefect.

You can see the time for 100 gettimeofday calls. Raw made with 100 copied macro calls.

Next you get 100 low level logs with a small payload.

Last the argv write 2, but that is a different story.

So you have a rough first time for the gettimeofday and log. And its in the region of 40 click on my box for the gettimeofday ( If you check the code you will see that a simple C function takes rough here 20 operations itself for the stack thing, the prefix code and the return suffix code. So its more 10 to 20 click for the time itself. Could make it inline directly in the writes one day ...) )

For the log a time of rough 100 clicks is OK.

The real cost come in for the transfers of the buffer. The problem is memcpy.

You can see the numbers when you convert a buffer.

Here for the sake of a proper discussion a run:

```
$ atrshmlogtest03 1 2048000 16
```

```
shm log check for c++ code and threads
start of threads.
dispatcher has count 1 and looptime 2048000
packages made
0 198423611
threads delivered futures 198423611 198423611
main 198467333
0.99978 1.00022
end of test.
```

I spare us for now snapshots.

Well, for starters.

The thing made 2048000 times a loop that makes a gettimeofday, a loop of 16 times int++ for a volatile operations – the compiler could else evtl do something against it – and then the next gettimeofday and the log.

Then the next iteration.

And it took 0,198 seconds (that's the German way, the , instead of a . ....)

2 million logs in less than a blink of an eye, and still traffic on the process...

The reader did its job.

```
$ atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 8
count of initial used writes is 24
enter transfer loop ...
logging stopped by signal 9999 9999 ...
logging done.
writer data : time 464597670 count 314 per use 1479610
```

That's a ssd below. An Intel 320 so we have good times for the 9 year old box with its sata II

interface.

We have a time of 1479610 clicks per write. So if we have the transfer times for the buffers we can start some simple calculations.

The convert gives us the times:

```
shm log converter from file 'd1/0/atrshmlog_p25812_t25823_f0.bin'
to file 'd1/0/atrshmlog_p25812_t25823_f0.t
xt'.
id    3 acquiretime      780 pid   25812 tid     25823 slavetime
2308860 readertime    1087060 payloadsize  5
24280 shmbuffer      0 filenumber      0
shm log converter from file 'd1/0/atrshmlog_p25812_t25823_f1.bin'
to file 'd1/0/atrshmlog_p25812_t25823_f1.t
xt'.
id    4 acquiretime      300 pid   25812 tid     25823 slavetime
472620 readertime    780540 payloadsize  5
24280 shmbuffer      0 filenumber      1
shm log converter from file 'd1/0/atrshmlog_p25812_t25823_f2.bin'
to file 'd1/0/atrshmlog_p25812_t25823_f2.t
xt'.
```

OK. Now we have some numbers.

First we get the acquiretime.

780 clicks. Well that's a slow one for static buffers. Next is 300, more in the right direction.

For an dynamic you can easy get a 50000 ...

For the slavetime we get an awful 2308860 here. Bad. The next is 472620 , and this is still far from the 200000 I get when its right. So much for same thing same time. You are far from it. You are in the land of memcpy on real machines for half a MB buffers.

Next is the reader for its transfer – should be the same right ? Same thing and that stuff...

For the first its half the time (HM, better) but for the second its twice the time. OK, that's for our first two here... later we get better times, but still the reader has a problem. So we have to ask if its for having too many threads – after all the test03 did it only with one thread.

The payload size shows we use full buffers here. So we have to check for small transfer times if it is simply a empty one.

The shmbuffer is the index in the area. This helps to detect if we have concurrent use of them –

numbers are changing – or only use one – numbers are the same for buffers.

Last is the file number, its also in the name but I have it separate.

Now let's see if we can make it better. We reduce the number of threads in the system.

```
$ export ATRSHMLOG_FETCH_COUNT=1
$ export ATRSHMLOG_WRITE_COUNT=2
$ atrshmlogreaderd d1
shm log attach and loop write file.
logsystem version is 1.
directory is d1
files_per_dir is 10000
stop via signal reader for pid 9999 with value 9999.
directory not found or not accessible 'd1'. Create it.
count of initial used fetchers is 1
count of initial used writes is 2
enter transferloop ...
logging stopped by signal 9999 9999 ...
logging done.
writer data : time 537103360 count 314 per use 1710520
```

So we got it this time with one fetcher and two writers – the write time is comparable.

Now the convert gives us

```
id      4 acquiretime      180 pid  27804 tid    27815 slavetime
      168160 readertime     227460 payloadsize  5
24280 shmbuffer   0 filenumber      19
shm log converter from file 'd1/0/atrshmlog_p27804_t27815_f20.bin'
to file 'd1/0/atrshmlog_p27804_t27815_f20
.txt'.
id      3 acquiretime      630 pid  27804 tid    27815 slavetime
      159190 readertime     229290 payloadsize  5
24280 shmbuffer   0 filenumber      20
```

To be honest, it takes some buffers to come there, but now we are in the same league for the two.

So much for using too many threads.

On a box with only two real CPU cores its difficult to make such things. On a big one you can see that the mt thing is needed when you get a high load. That's right for the slaves and the reader. But for a two or four core box you can start with a one. That does the job.

If you have a permanent log and you need rough 200000 click per transfer the total time is 400000 click in the area. For the 16 buffers I use here that mean we can get through in 400000 click 16 buffers – a least in theory.

This would mean I have to write down in 2000000 clicks – that's the average for the reader to make it for one buffer to write down – we have 5 times 16 buffers – that would be a need for 80 writers – and they would have to do it nearly without interference for it or my reader would explode in some seconds – even if I have 1024 buffers in place – there were 314 in just 0,198 secs ...

So much for the guys who want to log on a box with let's say 512 MB ram .... You are finished in just barely one second. BUMM.

Back to reality.

If you do the usual program logging it will be more than OK to have a simple setup.

If you have a real high log load – lets say an 20 GB application server with round about a thousand threads – that's different.

There is first the need for more buffers. Hopefully the thing makes good use of the turn off and stop for threads. Or if it is one of the start many threads in a short time then kill them things – you get it. You are out of memory for the thing in no time. And your box too.

For the reader its more a question to make it through the first seconds. If you have a reasonable fast file system you can expect an average write in about 1500000 clicks. That's about 0,5 milliseconds.

If you create an amount of buffers for let's say the load of the test03 in the first second it would mean to catch 1500 buffers in this second only. So you will likely need to raise the number of pre allocated buffers. There is an ALLOC ADVANCE thing in the reader for it. Give it a 5 – wups, that's alone 2.5 GB for the record – and make it real ram (it uses 1000 times the number of buffers ...). Using the swap space or page file is out of the question here.

So this is the way we can use it out of the box.

## **Now we know how fast it is – can we make it faster ?**

So you have it up and running and you have problems still to come up. Or your program is crawling.

If you have control for your application and the module there are some options.

Before you start to do anything else you first have to find out what are your needs.

To do that a small statistic run can do the job. See below for the details.

After you have the statistics you can add info that is not known to the module for now.

How many threads do log at a time ?

Answer: NumberA.

How many threads do I start ?

Answer: NumberB.

How many of them need log buffers ?

Answer : NumberC.

How many buffers do I get back after the threads finish ?

Answer: NumberD.

With that info you can start.

## A low throughput scenario

We start with a simple thing.

We have only partly logging. And its only a low traffic in the system.

When we check our program we see there is a small NumberA, perhaps only one . The mksh example is such a thing. The test00 to test02 are those things, also the defect.

So you have a low number of threads. And your NumberB is practical 0 – no additional threads.

You do logging for the thread, so the Number C is equal to Number A.

For the last number its none , zero, 0 . You simply end the program, the buffers are moved to the area by the cleanup. You do NOT give the buffers up after logging. Perhaps I should change that.

In this scenario we have no concern about the number of buffers in the active system. We start with the statics and we likely never need more than the buffers for one thread. This is 2 in the default. So we have 1 MB buffer memory in place for them. The rest is 31 MB and simply dead memory.

We do log, and we have for some of the programs that the run time is limited. So is the amount of log.

For the one program that does run in principle an undetermined time – the mksh – we have to accept the log can be undetermined too. For the others its limited to the amount of some entry's – best is about some hundred – and they are small.

This is a lot info, and we even have not started a statistical run.

So when we now try to make it better we have found that there are an bunch of programs that can live with the two log buffers completely.

OK. So we switch the slave off for them by setting the slave count to 0.

Next we can reduce the static allocated buffers. Set the define to 4 ,one reserve cannot hurt here , and change the static buffer initialization part.

This reduce start up time. And makes the memory footprint smaller. Less memory always ends up in faster, so we do it.

After this we have to start and run the thing with a reader – for now its reader d.

We know there is a bunch of threads to fetch buffers. But only a count of one or eventually two come in – so one thread would be enough. And a 0 is possible here . The reader make a final cleanup in the area, so if you insist you can wait till the program is done and then simply get the data from the area that's there in one final run.

Best is to simply do the program thing, and start the reader later. So there is no competition for the CPU and no lock contention in the area at all. We use the area as a buffer then the time we are not up with the reader.

Your program speed is now maximized. You get the log and this is done after the program has ended.

Not what you expected, but its for real. Try not to slow down a one shoot thing by making infinite loops in your mind.

Next is the mksh. Here we have an unlimited running.

## **A scenario for a long term running low throughput program**

You have a long term runner.

Meaning you don't know how long it will take. But its a low through put one. Say less than a buffer per second.

OK. We reduce again for the process the footprint by reducing the static buffers. We then try to make it with one slave – that's the default. So we simply not set that in the environment.

We reduce the thread CPU load. Its save to slow it down in the loop in the wait part. Put 1000000 in there. Its a run of the slave every milli. Perhaps even more.

So do that what will disturb the client as less as possible.

For the reader we have one fetcher. And we have one writer. That should do the job.

## **A scenario with low throughput and multiple threads**

This time the program starts threads.

Its still only a low throughput. So we have again only one slave. The adjustment of the static buffers cannot be done if you don't know the maximum thread number. So we leave this first. We have a reader with one fetcher and one writer.

Now comes the new part compared with the former scenario. The threads are started and stopped. So for logging threads its vital to make at the end an stop or you use the turn off.

If you don't do that you have a memory leak. And its a big one. One MB in default.

So your process should give up the logging in threads as soon as possible. And it should try to be sure of the stop or turn off for the threads. After the thread has been stopped its too late. No more access for the threads local possible, so no more cleanup for now.

## **A scenario with high throughput and small number of threads**

This is a case for checking you have still enough buffers static. A statistic run should show no use of the buffer allocation with dynamic memory. Or you check the id numbers – if they exceed the number of static buffers its sure you have dynamic once in place. If you can do that adjust the static buffer to cover your maximum.

Next is to check if you encounter many buffer waits.

This indicates you have a transfer problem. Check how many slaves are running. This should be a reasonable number – about one slave for 16 threads is a good start if your threads log much.

You have to check how many buffers will be at the same time in the area.

If you have the numbers from the converter you can see the numbers of the used buffer in shared memory. This is changing if you need more than one buffer at a time. If its not changing from transfer to transfer you stay put. Not changing means that more or less one slave thread works at a time, so you can reduce them and spare CPU time.

If you have more than one buffer in the area at the same time its time to raise the number of fetcher in the reader. Start with two.

Same goes then for the writers. The ratio of time for a transfer to shared memory plus time for a transfer into reader to average write for a buffer to the file system comes in.

You need enough writers to come up with the number of buffers your fetchers put in. Or you get a bloating reader that can slow down and then your area is simply full – slaves encounter a wait till the transfer can be done. Because the reader has a 1024 buffer queue for buffers this can be undetected if you only do some logging, you need at least about some thousand till a shortage is here a problem.

So you have a complete transport chain now and you must honor that.

Still most vital is the thread cleanup. You need to maintain a stable number of buffers. This is the most important thing to do.

## **A scenario with very high throughput**

You have high speed and high logging threads.

So you can get a throughput of number of area buffers in a time that is the sum of the time for copy to area and time to copy to reader.

This is the theoretical max. No way to do it faster. So you raise the number of buffers, but this will also raise the amount of memory used. And this is indeed a factor for the system speed – you have more memory than the second level caches of the CPU will cover, and likely more the third level cache support – so you slow down to conventional memory speed.

If you try to start more slaves be prepared it can only cost more CPU for you – they simply wait. So the max should be half the number of buffers in the area. Same for the fetchers. If the time is

comparable to the time for the slaves use also half the number of buffers in the area – that's the default for the reader anyway.

For the writers you need to make the same number of buffers you get in from the area now to the file system. Take the average time for the write and you can calculate how many writes you have to make parallel. And check this before you start – some file systems simply do not like parallel. And some hardware don't like it too.

When you test realize that you do most things in the first time in memory – buffers for the file system are cached and even disks have their caches and the reader has at least 1024 buffers in place..

So you have to test at least for some time to be sure – a minute in high load should do the job, so in the second and third it should become realistic.

Raise the number of static buffers if you need it.

Now the statistics come in. Use them for at least some test runs – you will need that info.

If you encounter still high counts of buffer full waits – switch the most important for speed to the discard strategy – its better to have a thread doing its job than waiting for logging in that scenario.

# Statistics

So you have it done but still not sure its right for the adjustments.

OK. We have some kind of statistics. The so called counters.

The thing is in now. The most is in the process done by atomics. For the stuff that would slow down the writes is in thread local. We can get them from a buffer with the converter. Add a third argument as filename. The number of the highest sequence is also now in on the output and filename. You get the best statistic from the one with the highest sequence.

What can we get ?

First of all we get simple counter numbers. No timings. So you have to accept there is no log of the log itself.

For the numbers most of them have to do with simple setting something – so you can check if you made that. There is no count for the getter things.

Can be quite interesting if you are in a big system, have multiple developers and somewhere someone decided to do a thing that kills you. And if you have not access to every bit of source – you get it. See the statistics for a test run and see if you were hit by someone.

For example a set strategy process. You simply use the default – its wait – and someone switches over to discard. That's it for you – you loose your log in a buffer full scenario.

See counter 75 and 76 ....

Other things are the counts of errors.

If you use the simple way of life you do most things with the log without checking return codes. That's OK for me. If you made it right it works and you don't have to check for every thing.

But from time to time you must be sure to be on the road, so you make a statistic run. See counters 25 or 26 for an example. If you encounter such things you can start to check for it. If you don't check return codes you simply don't know till the statistics show it. OR see the safeguard corrupted thing.

Have in mind that a log call can be faster than the java thing to make an enum from a return code – so checking codes can be more expensive than do logging here. So statistics it is.

And then there are the real things – log discard, waits and these things.

If you are in a high load scenario you have to know if your configuration is good. So you need those numbers.

The problem for now is to get them out. You need to call the get for it.

To analyze them I have a little helper. The script atrshmlogstat. You need the numbers in a special format to use it. See the test03 for this kind of stuff.

And check the appendix for the thing. Its the best info I have for you for now.

## **When I change it, what then ?**

After you have changed the module and it fits you have a simple question to answer – is that enough for me in the future too ?

It can be that you never plan to update it, so then your answer is yes. No changes needed any more.

But if you have to live with updates – then this is at least a possibility you must take care of.

## The Adjustment

We have the module in the vanilla version doing its job good. But if you need some different behavior it can be you have to do more than simply use the environment settings.

So we have to discuss here if you have to change the code.

### Changes in the first place in atrshmlog.h

You can get some things here.

Use of a different platform with special needs for example. You add the define, then switch the others off.

You duplicate the codes and then set them different.

For example you use a clang on Linux, not the gcc.

So you have to adjust the inline stuff and the assembler. Most likely this is not the same.

Other things can be the use of a different threading. Switch from a posix pthread to a platform thing.

Plenty of things can be done in atrshmlog.h then.

There is:

- use of inline code
- the used thread model
- the use of get syscall if available
- the used real time clock time
- the nanosleep
- use of the tid model
- default timing function for the gettimeofday
- some typedef stuff to cover system stuff like times
- helper types for pid, tid and such things
- all macro code – you can put something under cover in there if you need it

That is all in for the interface. So it makes a new module – even if you don't change the layout of the area its eventually worth a new one.

### Changes for internals

So you came on a thing I have not in for now.

Let's say you work on a critical one and need best performance. And you have a real big iron. If this is an option – set a bigger buffer size and you minimize the overhead.

OK. You are now in the internals, its this time atrshmlog\_internal.h.

You can change here a lot – and sometimes its not only here but also in the source then.

For a bigger buffer there is a define – simply use it.

If you have the opposite – for example your multiple core thing only support 256 k as second level cache – reduce it.

Recompile and that's it.

You can do a lot of things here.

For starters:

- change the buffer size
- change the number of preallocoed buffers
- change the shared memory access rights – if they are used
- change the default times for slaves and waits
- change the number of buffers in an dynamic allocate
- change the number of events
- change the name prefix and suffixes for the module
- change the ratio of clicktime and nanoseconds
- change the head size for the logging entry
- change the structs (ohh. But don't forget to know what you do in the first place...)
- change the number of buffers for a thread

So you have this with only doing it here – with the exception of the head size and the count of static buffers. For them you will need the code changes too.

## Changes for the code

You have needs that go beyond the define stuff in the headers.

Well, this can happen.

Let's say you want to have the thread statistics before I can do it.

So you change the buffers and the thread local to hold them.

Then you add code to use them in the impls files.

You change the reader to transfer them – after all you have round about 300 bytes in the header left – or use a special stat buffer.

You change the converter and then you get the statistics. Or make it to the area and have some new dumper tool ...

You have to change the code.

And yes, you can. Its all open source and you are free to do it. Simply fire up your favorite text editor, and then make it to your own module.

I am not against doing that.

It start to turn out that the trouble is if it not works or – worse – partly don't work.

This is the thing you have to keep in mind : we live in the world of atomics, fences and memory model order.

For the rest its plain C code, so you should have no problems if you have done some development. After all I have made some documentation about it, so you can dive in and check for it with this help.

Also a post card for me is an option. Pack your distro – please no more than 15 MB as zipped file – and sent it to me. If you can do – drop the doc and mksh stuff...

So in doubt split the tar and sent me the thing in multiple mails.

What would be first target – beside the stuff that must be maintained if you change the defines ?

First candidate is a converter that fits your needs.

Second a reader that transports different things.

Next is code to make something happen in the module you need there – for example a dynamic start of threads by a writer signal (that's on my list, but its not top priority ...).

You can change the code. Add new files in the impls.

## Local changes for your own system

Now you have made the changes – what's next ?

First there is a good change you have done it only for your own. So you do not plan to give them to others for now.

Then you should prepare for the unthinkable – you have to upgrade some time to a new version.

Can happen – and yes, if worse you have simply got that changed bastard of hacks from your former colleague and now are confronted with it .

What to do ?

On most platforms you have a system that can make patches. That's a thing of a diff and the patch program (see a Linux system for that).

You build a patch file – this is easy.

Then you test it – run against the version that has been altered. Then check its identical to the version you have now to support.

Next is to run the patch against the new one.

I had only one file for the implementation in the beginning. Now you have a crowded impls directory. For the path tool this is ideal – no lot of hungs and mismatches – simply one or two lines here and there ...

By the way : if you do this with diff and patch – don't make your life harder than you need – don't try to change things you don't have to like

- comments
- simple variable names
- indentations

This is simply counterproductive here. Use the original source and change it – yes, but only what you need.

Don't start to duplicate by having a commented former line in. That's a bad thing. Use your version control thing instead.

You should also try to make clean line changes – replace one, delete one and add one for one thing. Combining things only hurts when you have to change them again.

And one thing last : don't forget to change the version number if you change the layout in the area and buffers. Mixing multiple versions on a box should always lead to a run time error in attach at best.

## Patches

Did I say patches in the former ?

Now its time for patches from me.

I have to make versions of it.

Its clear that sometimes its not the layout affected, so its perhaps only a glitch or a minor change for some new stuff.

Then it comes to a patch version.

You can get the new version – OK, that's the best – or a patch if your build environment supports it.

With a patch you can go to the new version. And if you have made patches then chances are you don't interfere with the patch and you have simply the patch in without doing yours work again.

So patches will be there.

You will get a patch number – so you can see it in the code and if you have to check from the compiled module too.

Making patches can involve a minor number switch. Then you have new functionality, but still the old interface and binary layout.

For the rest its a patch to go from one version to the next.

## **Wouldn't it be nice to get this, too ?**

OK. So you have the shiny new thing and think you will have to put in in the light so others can use it too.

Then its time to make a distro – please no files with more than 15 MB size – split if needed to different mails.

And sent this to me.

I will see what I can do.

If you have a patch that's fine with me. I use the Linux diff and patch tool, so you should sent this or I need your patch tool too.

I have a little catalog here. Please be prepared to make the answers in your proposal – I need it at least in the mail.

Does it

- work ?
- have an impact for other functions – eventually a slow down ?
- need permanent maintenance ?

Stuff like that. A simple list is sufficient. And a can even accept the maintenance thing if its GOOD.

## **The glory details**

OK. You now have made the module, tested it, used it, made even the layers, made the adjustments and have everything in place for the use.

Now you have the luxury of some spare time and you want to know how it works. And you are for real this time.

So I try to give some info about the how it is done and the why it is done here.

We will first use the theory approach.

Then comes some more or less understandable explenation for the non C guys.

At last we dive into the C module and see the things for yourself.

## Theory of the module

We use here an analogy. It is far from perfect, but you will understand the most after that.

Imagine a house with a balcony at the first floor.

At the garden there is a small pool of water.

Its for some party thing and there are people in the garden and in the house.

Somebody screams, then something crashes. Again screams.

“Fire ! Fire !”.

There are some buckets around, and the guys in the garden run to the pool – there is plenty of space around and some guys start to put buckets in the pool, fill them up and then put them at a side of them.

Others take the full buckets and run to the balcony.

In between the guys at the pool grab the next empty bucket and fill again.

If you understood this – welcome to the client part of the log.

We use so called alternate buffers for the log. They are the buckets in the analogy. You fill them up with the log info.

The bucket has a maximum size. So when it is full you need to wait for it to return empty. Or use another one. If you are out of buckets – well you can ignore it, or wait....

The guys at the pool are your threads. They all use their own buckets. No one uses a bucket together with another one. So most of the time you work in logging in the thread and without any interaction with others.

Now to the guys that transport the buckets to the balcony.

This is the so called slave thing. Its realized a bit different, but you can think of it in this way. A thing to simply transport the full buffer to the shared memory. In the analogy the bucket to the balcony and then to the guy that stands there and grabs it.

Now we switch to the balcony thing.

There are standing some guys inside of it. They grab the bucket from the guy who takes it to them. Then they turn around and put it on the floor of the balcony entry. There is just enough space to do it with one bucket for each ( well, give it a try and think of big buckets here...).

Inside the house the people have to grab the bucket from the place, then move it to the fire and then do the thing.

In between the guys at the balcony move bucket for bucket in.

That's the transfer into the area. There is a limited number of buffers you make. The same thing for guys that can work on the balcony. If the place is limited you can only put a realistic number on it

or the thing will slow down itself.

So far we have the second part of the slave thing here. It transports the buffer into one of the area buffers.

To do this the area buffer must be free – imagine in the analogy that there is place for one bucket behind the guy on the balcony, so he simply cannot put multiple buckets in without first having to wait the place is free again.

In reality we can put the buffer at any place in the area as far as its empty.

Now comes the last thing. The buckets are transported inside the house. That's the thing we call the reader in the system – we transport the buffers there from the shared memory into the reader address space and then do what we have to do with them. So we clean the buffer in the area after the transport is done the first part of the way – we do not wait for it to reach its destination, its enough to have the space free for the next buffer.

The guys who transport the buckets in the house to the fire are a bit slower then the guys that simply take the buckets at the balcony entry – which is a small door or so - and then give it to the inside guys.

So in our system we have the fetchers who grab a full buffer in the area and transfer them in the process address space on a list. Then they free the area buffer for the next transfer by setting them back on the free list in the area.

OK, sometimes an analogy does not match exactly, but for now it works.

Last thing is the guy who took the bucket and runs to the fire and puts it down.

These are the writers in the reader. They take the next available buffer from the readers list of full buffers and writes it down to the file system.

Here is where our analogy ends – the buckets simply dematerialize and after that rematerialize at the pool. So much for a computers way to copy data.

What happens when the fire is stopped ?

Well, first of all all guys at the pool stop – but they put the half full buckets at least behind them.

Next is that the guys start to clean up the place. Somebody will put any bucket to a place where it is needed and then the rest of the water is used.

In our system that is the job of the cleanup. It checks for all buffers and puts the not cleaned up in the area at last in program end. So nothing is lost in a regular shutdown of the program.

On the balcony the last buckets are moved inside, then the guys there stop. So again the things are moved in.

For the transport in the house the things are then moved where they are needed. Again in the shutdown of the program there is a cleanup so any already fetched buffer is then written down to the file system. So nothing is lost here, too.

OK. That was for the basics. Nothing so far from C or Java or anything for a programmer.

Still we can get some insides from the analogy.

First of all – there is no unlimited in this game. We have to accept limits.

First we have the fact that a bucket cannot be too big – no one could handle it then.

Our buffers are limited. And its a hard size limit in the code. I choose a thing that works nice for a small number of threads to do things. It fits at least in the second level cache of modern CPU's and so its fast to be moved. Anything bigger and you have to test it out.

Next is the number of guys at the balcony – the space is limited. So is the number of buffers in the area. If you think of a slave as a transfer of buffer from the list of full buffers you simply see that it makes no sense to have more slaves than buffers in the area – they simply will start to wait to do the job if the load is high.

Next is the fact that we accept only one bucket for every guy on the balcony inside. So there is a limit too, and its the number of buffers in the area again. Having more fetchers to get buffers from the area is simply not useful, so this is again a simple thing we have to accept.

If you think of it – if they both need the same time for their thing, the place for a bucket is somehow used half the time from the balcony guy and half from the guy that fetches the bucket – so in practice already half the number of slaves and fetchers will cover a 100 % use of the area – any more and one of them starts to blockade others.

Because the operations are similar, there is in theory a one to one for the two. In reality it depends on the speed of the system as a whole and the additional write stuff for the readers what timings you get.

Another thing. Everybody has to take care not to do one of two things: working without respect for the other or waiting too long if there is the chance to do the thing.

So we have to synchronize the work at any place that the guys have to share a common thing.

This is done by using some internal handshaking protocols based on flags and by using some small time wait functions that use reasonable times for waiting. So the buffers cannot be used concurrently and the fact that one is using a common thing results in a short time wait for the others involved.

OK. If we have enough guys to do the thing – we have the maximum throughput of water.

Fill the bucket, transfer it to the balcony, put it through and in the end transfer its in the same time from the balcony inside the house. Of course the things in the house have to come up or we simply transport much more buckets than can be used in that time.

So in our system we have also to respect the time we need for the write down of the buffer into the file system. And we have to use as many writers as needed to come up with the fetchers and slaves.

That's for now with the programmer independent stuff. Now we get into more technical things.

## **The way to log in shared memory – or how to circumvent it**

When we try to understand the way of logging that is implemented we have to start round about four months after the beginning.

That time I had a nice running logging for the single thread system mksh, but it was not the best in case of high speed nonsense things like a raw counter loop.

And I had already started some multi threading things because it was clear the thing would make it after to the shell to a java logging in application server environments.

HM. Bad thing. Even the single thread mksh told me I was wrong with all the knowledge of the past 25 years in UNIX ipc....

The timing was at best clumsy. Real time marks with about 500 NS to make them. The write of a log was fast, but I had those mutex things all around and when I tried to scale up – more or less a blockade from everyone against everyone else.

So I stopped twisting some simple C transfer things and made some bruit force experiments with the shared memory use – random, sequential, cross bar, interleaving ....

Nothing helped.

It was time to take a break.

Lean back, close your eyes, and start to think over.

Question: What is the best way to log the data ?

Answer: I put it with highest speed into a buffer that I control and nobody else.

Question: What happens when the thing is full ?

Answer: I put it on a full list and go on with my work.

Question: What when my buffer is on the full list and I need to log ?

Answer: Use two buffers alternatingly.

Well, that helped.

Make a small dummy and see how fast you can put log info in a local buffer.

Next, how to make a list for buffers so I could put these on the list when full and use the others.

The things began to work again. I could do rough 16000 log entries without interfering with anyone. Only the timing thing was bad.

I knew from my time at the University of Cologne how super computers did the timing – with a special register thing called a tick counter.

So I gave it a try, and bingo. The question of making time was reduced to 20 nanoseconds and I had no longer a system call.

Making a multithreaded logging with a support thread is not so tricky, if you do the list stuff by the

book. So I took examples from the net for the posix way of live. My audience was in these days the posix and C99 community.

When I came to the atomics it was a bit too early – I had read about them in the working of ISO C11 and C++11 stuff, but I had no practice that days. So I switched on that things and started to replace the posix mutex and condition variable stuff.

After some weeks of more or less nice try's I realized that this was serious stuff – but there was no real help in place, no article on the net and worse no book about the C11 way.

I checked the PDF about multithreading in c from the Linux kernel guys, but that did things different.

So I made a move to Anthony Williams book for C++ and checked the stuff.

After having a rough time in making slow progress I could start to remove more and more of the not needed things – and I had a lot of them – and finally got the atomics things right in place.

Now the synchronization was smooth, and I had at least no longer the need for posix in that area. I began to make progress also for the area synchronization. Now the multithread reader began to make sense.

My calculations for throughput and parallelism came near the reality for my big boxes – two AMD Opteron Boxes with 24 and 32 cores and plenty of ram to be used for Oracle Databases and big application servers in development.

To have a scaling by more than 90 % was still tough, but possible as long as I had the number of CPU's in place. Sad to say that some operations simply took three times longer than my box – a P9600 at 2,6 GHz and 8 GB notebook from 2009. So much for AMD Servers today.

So to summarize this: doing logging into the area directly is the first thing, but it will end up in more or less heavy interaction of the threads via the synchronization stuff. Forget about this.

Next you have to accept is that logging in the program has to be made on thread base – no central logger stuff – same thing as for the area. Forget a central buffer system.

Next is that you will need fast timings – and I mean here for a log in nanosecond league no more than 50 nanos in the first place. That's already dangerous near the 70 nanos I need for the log entry to make with a 20 byte payload – if the buffer is not full.

And of course you need more than one buffer per thread. At best two. You have to accept that the buffer has to be moved by somebody else – not your working thread – and that takes easy 200000 to 700000 CPU ticks for the half MB I use. You can't wait that time or do it by the working thread itself. You NEED slave threads to do that move and the alternate buffer to log on in that time.

OK. So far we are now again on course, and that is exactly what this log is. We are still on course now for the theoretical best log possible. Any thing different needs at last a technological base for the memory transfers – perhaps I switch to transactional memory later on, but for now we use only the buffer and shared memory thing here. So we have to move the log buffer after it is full.

For the list stuff – its a lie. Today I use an adaptation of the stack atomics demo of Williams at all places. Its after all not of concern when a buffer arrives the file system, so I can live with an upside down order in writings and moves all over the place. Important is that the stuff works and is fast enough to come up with the 60 to 70 nanos a write takes – forget using condition variables or mutex's here. We need synchronizations in the 10 to 20 nanos region.

So after all I say, its clear that we MUST end up with this design. And for the details it can be made better, but for the design I have no alternative.

Points:

- We have to log in internal buffers only our thread uses.
- We have to log to multiple buffers in a way we are not stopped by full buffers.
- We need some background thread to do the real transfer from buffer to shared memory.
- We need a fast as possible timing function.
- We need as low as possible the cost of synchronization and if possible no blockade.

That's all for the moment. The rest will come clear if you check the implementations.

Next we have to clear our understanding of the atomics and the memory model.

So I use an analogy here, that will do the job and we don't have to check specs of a CPU vendor for the stuff.

## Cindy's classroom

We enter Cindy classroom. She is a teacher at a school I have forgot the name for now.

There is her desk near the entrance, the table in her back, and then the rows of tables start and line up till its the end of the room.

Not very many, about 20 children or so.

The first row is desk by desk, the next have a small passage so the child's can reach the desk without having to go parallel. Only for the entrance to pass, go aside to the end, make it to the last row and go back between the desks in the column.

Well that's a bid odd first, but let it be for the time a fact. Its normally not the first row dense closed, but here it is.

First day of a new class, and so she puts some sheets on the desks.

The children arrive and the first thing after good morning is a simple task.

Write down ten numbers on the sheets for the first row guys. Put your name on it too. Number them from one to ten.

Last row get the task to ask for a desk and a number by the index and the thing is then copied on a sheet and the children have to put the sheet to the right desk in a shortest distance way through the lines. No crossing, no paper planes and no snowball things allowed.

After the things are back the first row has to announce the results and Cindy checks them.

Everything is OK, and for today the class closed.

Second day.

This time the game was changed by Cindy. All children in the last row asked one by one for the numbers, and the in between had to make as few transportation as possible, its now n time. So they started to combine desk and number combinations, and after the last row did the job the things came back.

This time there was a surprise.

Cindy : "Well, this is odd. Paul, can you explain the 56 here ?"

Paul took his notes about the numbers he had made and – it was right – he really got them.

Cindy: "HM, Mark, can you give me your numbers ?"

And so Cindy started to write down at the table the numbers and the flow of them.

After the third she came to Sally's notes.

Cindy: "I'm a bit puzzled – why did you gave Bob's 37 in here ?"

Sally: "I got Mikes result and there was the 37 ."

Cindy : "Oh, that was a bit unexpected by me – you didn't wait till it was given back to you from the second row ?"

Sally : "No, of course not. I gave him the number as fast as I could – this was after all what we was told to do !"

Cindy : "Ah, I see, for now its my problem then."

And again the class was closed.

Afternoon this day:

Cindy went to Tom in the corner of the teachers room. Tom was the oldest of the teachers, and he had the most reputation when it came to giving support. She told him her problem with the class and the way they did the thing. Tom blinked once after his glasses and said : " Come."

He went with her to the second floor. In the teachers room they met Tony, the science guy.

Cindy told him the thing.

Tony: " Oh, nice, seems you have a cache consistency problem."

Cindy : " A what ?"

Tony : "Same thing as for our today computers. You need rules that the children have to obey for the transport and the age of the calculated numbers or you will be lost."

Cindy : "?"

Tony: " Let's start by this. You name from the last two desks one time the first desk first number..."

Cindy: " That's Tim's."

Tony : " .. ah jepp, Tim. And then the second desk first one too."

Cindy: "OK, Tim first and Mike first."

Tony: "Good, then the second desk first again, and the third desk first."

Cindy: "HM, can we make a picture ?"

Tony: "OK. I will do it."

And the picture was made.

Tony: "So we start here with Tim first, a 12, Mike first a 23 and Bruce first a 7."

Tony: "Now we make the sheets her and route them to the desks of Hilda and Marian."

Tony : "Hilda calculates the new 35 for Mike, and for Tim."

Tony: "And Marian does the 30 for Mike and Bruce."

Tony: "So far so good – clear ?"

Cindy: "Sure. But what ..."

Tony : "Wait. For now we have made copies of the numbers, and there is nothing to give any number a preference over others, so they are made equal. Now we enter the caching game. We route them back and someone wants to calculate with Mike's number."

Cindy: "But I never .. OH. There are now two of them..."

Tony: "Yes. You never told them to do it, but they simply routed the 35 and 30 directly back. And worse the original is still in place."

Tony: "And now Denise and Theresa did the calculations and you got 42 and 68."

Cindy: "So the problem was that they didn't wait for the numbers getting back to the first row."

Tony : "Exactly. When they wait its a lot of speed they loose, so they tried to do the best to make it as fast as possible. And so you end up in calculations that are made on the intermediate results, not the originals in the first row. That's what we call the caching problem."

Cindy: "OK, now I see that. What's the solution ?"

Tony: "To make a long story short : everything you like from an announcement to all to a grin and ignore it."

Cindy : "WHAT ? Really ?"

Tony: "Yes. The computer vendors have made different solutions, and this is one of their keys in business to make it better then their competitors. Or at least they say so after making a test drive looking good for them."

Cindy: "Any simple and not so simple example ?"

Tony: "For starters: the announcement. Every time a child needs a number it announces that and every child having it names the number and its time when it was made."

Cindy: "So I need now the time too?"

Tony: " Yes. No way without it. The youngest result wins. All have to replace their result with the youngest one and the show goes on."

Cindy: "That will give a lot of noise ..."

Tony: " Yes. And so it will do the job, but it will be a slow and rough approach. But it works. Next is the buddy's go alone thing. You have to calculate for the desk column only. So no numbers can go from column to column , only in the first row its allowed to change a column."

Cindy:" Let's see .... ah, I see now."

Tony: "That's right. Its a bit better and you do not have the same problem in the column."

Cindy: "But still when I come to this column here and that there I go wrong."

Tony: " Yes, and you can still have another problem, that's the cache line buddy's. To make that you need to use a second number together with the first. Then you make some calculation for the second and ask after that in two different columns for it. After you have copied them also with calculations

for the first number things get pretty rough.”

Cindy: “But that's again ... wrong.”

Tony: “No, its two times right but from different points of view. So you have again a time and compare thing to help you, but this time its the whole group of numbers on a sheet. That's the cache line.”

Cindy: “OK. I see where this ends. I will never get the numbers right if I don't give strict orders how to handle the order of things.”

Tony: “Exactly. You need at least a time and a clear rule how to handle the thing for the desk itself. Then you can go down the desks in the room and solve it in one or the other way. At the end you come up with a bunch of rules and that will determine the results. “

Cindy: “OK, That's enough. I have to start in five minutes and must go now.”

On the way she began to make the rules for the third day.

Third day:

And the class got the correct results.

End of the analogy.

## Back to work

So we need to have cache lines in mind. Having numbers in a block of memory that is moved from cache to cache inside the CPU. And for different levels of caches. And pages. Moved from third or even fourth level caches to others.

And cross bar and numa architectures. And store queues for delivering newest values ...

Oh boy.

Now we have to enter the multithread and multi processor game, and the only thing that helps it to make use of the new C11 and C++11 memory model thing.

Anything before is at best dependent for the platform in use and a black art of programming only told in classes that seemed never reached the light.

We start with the memory model.

Same for both worlds. There is a set of rules to handle that inside a program – and that means strictly that we have no ground if we leave the program and do the shared memory thing. But at least for my box it works and for my big once too. So I think its at least save on all posix systems with the Intel architecture.

We have to accept that the memory model has two ways to handle things.

First is to play safe. Use the consistency model and that its. Any intermediates inside the program memory will synchronize before you use them, and that's independent from your platform.

Second way it to use the release, acquire and consume relationships.

To make it short, the module does that second thing. All over the place you will find the acquire and release things. And some consumes.

All is tested. In the beginning I found the need for fences – the term is sometimes known as memory barrier – and did that too. But in the end I got the dependency's right and the release and acquire synchronized as expected after the things Williams told me. Good.

So I removed the fences, but then I realized that I could have at least problem on platforms not doing it like the Intels and AMD's. So I did make a compromise : having the fence still in but also a flag to ignore it in the default version of the module. Later on a getter and setter. So you can switch in theory on a power or sparc or alpha and check for the thing. If you need them you switch the fences on. When it works then change the default settings in the module and make this with a platform define depending on the platform – and sent me a post card with it, I will integrate it then.

The other thing was the atomics stuff.

After playing with the stack example and checking for the queue I switched to the stack. Its simple and if you do it by the book you end in a fast and simple construct for the pop and push operations. The rest is simply to decide how many of that things are needed and be happy if they work.

The queue has a bit better performance in case you have permanent concurrency, but this is

normally not the case for the module – we simply do seldom the things that can hurt, and we have nano sleep all over the place, so chances are slim that we really hit another thread here – but its done and will work then. So much for the theory.

OK. So we have now atomics.

That gave an unseen draw back: In C and C++ the things work the same and the C implementation could have been the backbone here – but it didn't. So I had in my interface and its basic structure now a construct that had the same name and worked different for C and C++. Worse the signatures were different too. The C++ compile failed.

So I moved everything to the internal header. Cleared every use of an area struct and made void \* intermediates.

Now the C++ does simply not know the layout of the C structs, and I can live with that.

Still plan a C++ version in one of the next versions to support those platforms not using C11....

OK.

You can check the internet, I suggest you take [cppreference.com](http://cppreference.com) and see for the atomics yourself. For the fences its easier in practice if you use Williams. The way the committee tried to explain it is at least not very transparent.

After having done at least two versions where the things really went wrong even on Intel and AMD I can simply say its a good thing to have it work now. Don't spoil it if you try to use a new platform.

So after the basic architecture became working on real programs and even did it on the shared memory it was time to start the java layer.

This is perhaps a bit odd, but for the C module I see that we can use it in this way with our programs, but for the java module I suspect we will have still some changes here.

The thing is a full fletched layer, meaning I didn't left anything behind. Even added two helpers that you don't need in the C world to access the memory of the area manually.

OK. For this thing you have simply to check the documentation and you will find that it is a more or less one to one use of the C module function. So only the simplest thing that was needed from the java side was put in – the creation of new Strings for the returns of strings from the environment. All things else are made with the need of speed. No copy where I could come up with a different thing. In the end its a simple adaptation, and it took less than a week to become the version I have now.

There was some flow back of course. First the thing to circumvent jni calls at all cost.

The write became an internal fetch for the time. That covers most uses, the Point in Time and the Interval in time. Sometimes its needed to different timings and then decide, but if that is not the case you can now reduce the number of jni in Point in time to one and to two in Interval in time.

The other thing was the control over thread thing.

First I had only a few thread specific things and they were only internals. Now I had functions to inspect the thread for its internals and to stop, flush and turn off of them. And a bunch off functions to control that.

HM, makes the layer a bit more usable in the big run. So I accepted both and made the things in the C module and then in the layer.

After some silly playing I even did the things to make it a full fledged replacement – the create, the delete, the init and cleanup. Also the reads .

So now we have the java way to do things too – if somebody will do the coding for the support programs. At least a converter would be nice – the C version can handle the java string thing, but only partly, I would say at best its ill.

Back to the layer. Beside the flow back it gave me chance to check for the runtime dependancy and it was OK in Linux, but for fenster;plural things again got bad. Cygwin didn't make it, so the mingw port was needed. And I did it in three days. Its a bit more clumsy, and I doubt the timing will be nice with all that problems, but its at least the best we have – there is no other thing that works for now.

For a native port we still need first to switch to C++ - there is simply no support for C in the world of Redmond's thinking.

So for now I have set this on the list, but other things come first.

## A word about fences – or should I say memory barriers ?

Life is a bitch.

You just got it through the compiler, then you make a simple test run and all is fine.

Then you take a test03 and make a simple single thread run. All OK.

Then a test for 4 to 8 threads.

The converter blows off. BANG.

Whats happening ?

The problem was persistent, so I had to check it.

First I ran into the fact that the converter was right. The careful made printf stuff gave me false numbers. Was the first part of a buffer. The rest was clean.

Next I checked the bin file – after all it was the first part – and bingo. The thing was corrupt.

Next was to check the reader. Again it was corrupt – from the moment I got it from the area.

That was bad. No magic sim sala bim to make it fixed. The thing happened inside the program that was logging.

At least from my point of view that day. Now I am not so sure, there is at least a synchronization issue possible for the fenster;plural if you play games with some flags . Its an OPTIMIZATION from the msdn web side to make the transfer from different program attached to an mapped file – well, inconsistent if you want ... Never will understand what that optimization brings if your shared memory thing is inconsistent. But that's the way managers seem to think in Redmond. Sell shit and call it an optimization... should I next time transfer their bonus of the year by a system using an optimized but unrelayable mappe dfile thing and declare ater it crashed the boni that I was only doing what the msdn claimed to be an optimiz.... Fired at best same second.

So I checked next the logging itself. No error. Correct data till the buffer was dispatched.

Next I checked the slave – still correct in the buffer ...

And bang. A small check sum test made it clear, I had a different content.

This was a clear problem of the synchronization between threads.

As always I have buy'd the ISO PDF files. So I spent every three to five years about 600 bucks on the PDF files. I checked them carefully and found the fences for this kind of problem a solution. Strange was after I had read the thing that they had to be combined with a atomic operation. But that's the ISO committee thing. For Williams and his book things were different. Its possible to use them stand alone. Well, so much for committee work – did I already mention that Williams was the member for the whole multithread thing in the C++ 11 committee ?

OK. I tried it the ISO way and found it worked. So I had my first fence in place .

Now its time to say that the thing is better known as an memory barrier. It is some kind of fence too.

It triggers in last consequence an synchronization for the memory you have written or will read.

So I still cannot follow the ISO way, but for now I will simply try to use them in the way both describe it – even if I don't have the atomic in place.

When the module evolved I could reduce that stuff. Careful reading for Williams and transforming the C++ view of things to C11 leaded to a fence free system.

That's the good news: On my Intel and AMD box it can be made - nearly – without them.

Nearly – if I try something special it could be needed even here – so much for the low level thing, the lfence and that stuff in the Intel mnemonics web side stuff....

OK. So I am lucky for now not to use the set of operations that need them even in Intel land.

This in mind I decided to check for others and found that they indeed need the stuff.

Bad news. When I switch to let's say Power the game is reset.

So I did not remove them. I made for every fence – sometimes for some of them that are used in correspondence – a flag switch.

So its possible to switch them on if you need the, and I honestly don't know if this will be the case. Have for now only the Intel and AMD stuff so I can't speak for MIPS, Alpha, Power, Sparc or ARM now.

So here you get the now 13 fences . I say were it is – you could grep of course – and which could be a symptom. Then its up to you to try it in case you have trouble.

Why not on in the first place ? Well, its easy. You will encounter heavy memory synchronization if you switch them on eventually. And this can slow down a system dramatically.

For an impression: If you synchronize numa regions in an AMD box by the bios for may servers and try to make a super big – more than the numa region size – Oracle DB instance and do heavy insert stuff the thing can slow down to rough 1 / 30 of a one CPU system on one region. So much for dumphead administrators with no clue of hardware...

Be warned. If you don't need them – switch them off.

And here it comes.

## **Fence 1**

atrshmlog\_dispatch\_buffer

Symptom:

Your log buffer is inconsistent in the slave thread.

The list does not work.

## **Fence 2**

atrshmlog\_write0, 1 and 2

Symptom:

The log buffer is filled incorrect for some entries.

### **Fence 3**

atrshmlog\_read

Symptom:

The log is OK in the slave thread but your reader get a wrong check sum.

### **Fence 4**

atrshmlog\_read\_fetch

Symptom:

The log is OK in the slave thread but your reader get a wrong check sum.

### **Fence 5**

atrshmlog\_transfer\_mem\_to\_shm

Symptom:

The log is OK in the thread, the log in the area is corrupt.

### **Fence 6**

atrshmlog\_f\_list\_buffer\_slave\_proc

Symptom:

The buffer is OK in the thread, but corrupted in the slave.

### **Fence 7**

atrshmlog\_f\_list\_buffer\_slave\_proc

Symptom:

The buffer is not empty in the thread, but was emptied I the slave.

### **Fence 8**

atrshmlog\_alloc

Symptom:

Your allocated buffers seem not to be OK in the thread.

### **Fence 9**

atrshmlog\_init\_thread\_local

Symptom:

You area values are incorrect.

### **Fence 10**

atrshmlog\_exit\_cleanup

Symptom:

The cleanup does not deliver all buffers.

### **Fence 11**

atrshmlog\_cleanup\_locks

Symptom:

Your resources are not freed.

### **Fence 12**

atrshmlog\_init\_shm\_log

Symptom:

A fresh initialized area is still inconsistent in a following verify from another program.

### **Fence 13**

atrshmlog\_turn\_logging\_off

Symptom:

The tread you turned off still is running more than expected.

There are some fences in the readers. I simply leave them in.

For the 13 fences above its save to switch them on that are not in the logging functions. This will slow down, but if you have a sync every 5 to 10 thousand logs its not so dramatic. IF its in the write its different. Switch those on and make some tests if you really need them.

You have mno at least a helper for tests – the checksum. Its made in the client thread if you switch it on – and its slow and primitiv. Every transfer into a new thread it is recalculated and a counter hit if it is different. So you can identify a transfer problem at least this way. And check if the fence counters the thing. And sent me a postcard if it really works – on the x86 systems I got so far I did not need it any more.

## And now for the gallery – the C module way

OK. Now we are alone and we can speak from C programmer to C programmer.

This means no more hold back.

We will inspect the thing and have my short info about it. So you can decide for yourself if you can use it this way.

We start with the header.

Switch to a BASEDIR/src and start the favorite editor. Load the atrshmlog.h and we simply move from thing to thing.

The usual comments things. Then in line 57 the start for the platform defines.

I use four things of info here. The OS, the major architecture, some minor stuff if there and the compiler.

So we can support multiple compilers. And indeed we have the cygwin and its cross compiler for mingw.

For now the number of platforms is small at best.

The important new thing in 1.1.0 is that you no longer set a define here. You support instead the one you need in the platform script parts as a define flag for the compilers. So no longer tons of headers for platforms. Simply tons of dot files for the platforms ...

Then we get the inlineing. That's a thing you have to think about. You use inlineing not only for the C but also for the C++ compiler. So if you are lucky you can use it here. If your C++ complains about it you have to switch off.

For now I inline the time getter functions. That's a bit of a start, but no big ones after all. So the effect is small, but you have still the choice.

The next is the threading. It is set up to use the platform as the major steering info, but you can set it there to the thing you need. Its possible a thing for the future, for now the things are simply set by the platform.

After the Linux thing again for getting the syscall.

Then we have the clocks.

For the real time clock its the best I can do. So the platform is again in lead here.

Next the nano sleep.

That's a disaster for the guys for fenster;plural. See the internet for the things. Only millisecond. Period.

Then the thread id. OK. Linux is lucky here to have at least the syscall. The rest is the usual, take the posix and be happy. Mingw has of course its own.

I don't know if that is a problem, at least for the logging the tid is used as additional info for the whole buffer, not the entry. Hope we have no thread id mutation anywhere.

So at last the timing functions, and some macro. If your platform supports a better one you can change it.

First the idea to have it capsuled seems to be a good idea, but then the timing made it clear to have it more or less directly in a macro. So for now I have still the get\_clocktime here, but it is on the list of not needed features for me.

So we hit now the one and only include I need.

Stdint.h is for the basic types here. A 32 bit int, a 64 bit uint and perhaps a signed char or 8 bit signed int. I use the signed char for now.

That's all for it. All other things are expressed in int for the interface or a void\*. All others are done in the int32 and the unit types.

What follows re the basic types. I use here some typedef's, could made defines but I think we can use the typedef's here. So its the bunch of types we need.

Not much :

- atrshmlog\_time\_seconds\_t
  - The seconds of a time stamp.
- atrshmlog\_time\_nanoseconds\_t
  - The nanoseconds.
- atrshmlog\_internal\_time\_t
  - The real time helper.
- atrshmlog\_time\_t
  - The click time.
- atrshmlog\_pid\_t
  - The pid – and yes, the IBM has already made it in 64 bit land.
- atrshmlog\_tid\_t
  - The tid, also in 64 bit land.
- atrshmlog\_int32\_t
  - Our standard int type.

- atrshmlog\_event\_t  
The event flag – 8 bits are enough for now.
- atrshmlog\_ret\_t  
Our return type, again the int of the system.

Next we start to use them.

We have the event lock pointer, that is used in macro stuff. So I have to know it here.

Then the logging for the process flag, again needed for the macro stuff.

One last time the clock id – perhaps we get rid of that soon.

Now we get some nice enum stuff. The statistics counter, the error codes and the strategy.

For statistics its easy. Everything that is interesting can make a statistic counter click. So you can check for that. The price is performance and thread locals. See later.

For the errors its easy to use enum's and so I did it with a scheme in mind. Having positive is minor, negative fatal and 0 is the OK. Tried to use a tenth range per function and it seems I could nearly make it.

Strategy is best when it comes to the write code, so we spare it here.

So we enter now the macros.

And yes, I know most people hate it. But for this thing the macros have two uses, one is to make the test for the events and logging flag not too bad. And the other is to make at least some things possible that can be made with inline's, but then you have to expose code you have otherwise in the implementation header.

So I covered every function with a proper macro and played all the time the game of hide and seek.

At last for gettime it was worth it. For the rest its for now only some layer above. So I use the macros most of the time in the external source and not in the module itself – see for the program sources.

Only the gettime and the writes are of interest here – these use the macro for real and cover the event flag check with it. So the use of the macro is really with additional functionality here.

All other are for now only layers on top of the function.

And so after the macros we get the functions. And for the conventional functions that's it. So we can skip this too.

The real fun part starts with the inline functions.

Switch to ATRSHMLOG\_INLINE\_TSC\_CODE .

OK. Here we are back in the click time things.

The Wikipedia for tsc tick counter is the heart for the thing, so don't miss it. We have for now the only thing here for the gnu compiler and for the Intel way, but when the time is right I plan to do it at least for the alpha ( have one in my museum) the MIPS (also a sgi one here) the itanium 2 ( well, you know: there is a itanum2 here too), and a hppa 64 bit (also one of those in the museum).

Ahem, yes, of course also for real living ones. The Power, the sparc and the ARM – don't know if the MIPS is a real living one, its after all an old school sgi workstation. Perhaps I have to buy a MIPS board...

So for the Power its possible to rent one at IBM, and its for free for a month, so I hope to make it.

For the sparc I don't know. That will come later.

For the ARM I guess a smartphone will do it today. Have to learn android then....

So this is the road after the big three missing Linux flavors and the BSD things. Perhaps I make a Solaris x86 too.

Leaves only the Mac OS X on Intel – have at least a 2005 two core machine here for Power 5, but that's a bit dead on the market now.

So the Intel macs will come about two months later..

So for now we have here only the gcc and Intel stuff. Later I hope to have the rest.

After the clock thing we are through.

Wasn't so bad after all. No big structs, no self reference pseudo OOP with function pointers and all that stuff you see all over the place now with tons of macros for pseudo names.

Lets switch now to the internal.h, its the only one left.

OK, here we are atrshmlog\_internal.h

The first things as usual, the comment stuff and the includes.

Next is version. Please follow my guideline and change at least the version if you make a change to the structure of area or log buffer. So this is more a safety for yourself not to run in a version problem for the tools. If you plan to have multiple in place.

Next comes the usual bunch of magic defines that makes the thing what it is. So we are entering the adjustment.

First is the maximum buffer size.

If it comes to CPU s the second level cache is about in MB size, and the third level is – if existing – a some MB thing. Exceptions are the big ones, the VERY expensive big ones and so I use a size that fits for a simple process nice in the second and third level cache.

Make it bigger and you can slow down for the second level. Make it smaller and you spare perhaps enough to make it into two in. But I think that's at least a start.

Next is a thing I need for several places. The number of buffers we have in advance. If you touch it

you have to do some additional work in the module for the static buffers. See below.

And yes, its for 32 threads enough for now, so simple multithreaded things are happy with it.

The mincount and maxcount in the area. Use them simply if you need. But I doubt you have to change them.

Next are the rights to the shared memory and this is set for total access. Simply change if you need.

Now the time we wait in a slave when nothing to do. This is a bit in discussion now after the mingw disaster. Perhaps we have to set it in future for mingw to one million to get at least a millisecond sleep there. For now let it so.

Same for the next time constants.

Next interesting values are the preallocs. If you need a lot more threads buffer than the 64 from static you will need to alloc dynamic. And so this can reduce the number of real alloc's. We do it for that many buffers in one low level alloc.

Bad side: on cygwin and mingw this is memory we need to initialize for the time penalty of write. See below the attach for it.

Follows a bunch of things we have to use as limits in the initialization. We can reduce the buffer size, so there is a minimum.

We can set wait times. And we can set the number of slave threads to use to transfer to shared memory. And yes, even a 0 is allowed. See the adjustment for a scenario with that.

Next is clock stuff.

And now again an interesting one. The events max. We use a char buffer for the flags, but only one bit is used for now. So be sure if you make it bigger. You will need more memory then. Its helpful if you plan to instrumentalize a big system. Then you can definitely use a bigger one.

Next is the number of buffers a thread uses – to be honest I still have not checked if more buffers work better, so leave it for now to 2, but I think it will work with bigger numbers. You can try at least and if it does write me a post card...

The constants for the logging type. If you have an exotic environment with non ASCII you need to do the hard code thing like in the java layer.

What follows in platform stuff – the includes and some basic defines we need in the code, also some typedef stuff.

The depending things for threads follow, so you can see what it needs for now to honor win and pthread and c11 here.

After the tid's things we reach the default c stuff includes.

We need time and errno and ctype and stdlib and – ups- again stdint here.

Don't followed the development. Can be something is now no longer needed here.... Perhaps I

spare some time on docs and try to reduce here...

For now we are through after the now needed atomics.

OK. That's already 34 % of the thing. Defines, includes and the depending typedef stuff.

Now some defines that depend on the typedef's and structs are here. Mainly the head size of the log entry.

Next is the check part for the area – and a nice looking getter for it.

From there on we are in the area of initialization – the names for the environment and flag file lookup. We use a prefix and you can change it here. For the suffixes I say you should not do that, but its again up to you.

Last thing in defines are some hard code values and the Debugging flag – we have some printf stuff in place and you can switch it to on to see that when it comes to testings. But you have then also stdio.h in place and be sure not to corrupt your programs with it.

Next is the one and only pointer for an area. For now I don't plan to do a handle thing and use multiple. Its simply out of question for the needs of threads, could be done in about two weeks or so, but I doubt its worth the work for now.

Now we enter the real heart of the module. The structs for the area, the internal log buffers and some thread local parts.

First thing first.

After a helper for states of a log buffer we have the thing. Now its time to do this one by one.

## **The log buffer**

And here we are.

### **state**

We have her the first time an atomic. Its her to make the synchronization for the thing work. We simply read and write it in a atomic way and so we synchronize also the access. See Williams for that stuff.

And that's already all for atomics here. The rest is conventional stuff.

### **shmsize**

We use a size her. Its our buffers size.

### **next\_append**

Then we have some odd looking ints. In practice this is the shared memory analogous to pointers –

to be precise to ptrdiff's.

We cannot use real pointers. That's a fact for all shared memory systems I came over in the last 25 years. Forget it. Use offset's. And if you can live with it ints do just fine. If you think you need them you can use ptrdiff's. But for my little logging area the int is OK.

So we have here two lists. One is for the available buffers. They are free and you can use them next if you need a buffer.

### ***next\_full***

The other is the full list. You have a buffer on one of them. And the module has them in shared memory – so be sure you understand the memory model and make clear this works for your platform.

If you have any doubt you have to add the fence support. For now I know it works on Linux. For cygwin and mingw I have no negative, but I also do not test these too rough. So perhaps we have here a to do. But for Linux I am pretty sure its OK now.

### ***pid***

After the lists we have the pid. Its OK to have it here – we support multi process logging so we need it.

### ***tid***

Next the tid. Same thing. We need it.

### ***inittime***

For the next three I have to explain a thing. We use click times in the module. So you can calculate the difference and its nice, but sometimes guys want to have human things like milli seconds and that stuff. So we have the need to calculate these. Doing it from the actual click time is possible if you know the basic of your OS vendor, but its tricky if the vendor does not like them – You got it. Its fenster;plural again.

See for the Wikipedia and the discussion about it for them...

### ***inittimetsc\_before inittimetsc\_after***

So back to that. We have in the attach a real time and two click times. One before and the other after. So we can build a 50 to 50 match and say that is our start click for that real time.

### ***Lasttime lasttimetsc\_before lasttimetsc\_after***

For the buffer we have again a triple, and so we can calculate a linear approximation for the real times – its a real easy thing if you can use a simple linear thing. Only problem is you need more than 64 bit to calculate it. But that's stuff for the converter. For the module and its buffer those six values are the backbone to calculate the real time.

## **starttransfer**

The next time is the clicktime for the start of the transfer - together with the lasttimetsc\_before we can calculate the clicks it needs to get a buffer from the main memory to shared memory – so if you remember the introduction of this chapter – we need it to have a clue about the time a shared memory area buffer is in use from one of the copy things, here the slave.

## **acquiretime**

The next time is a bit simpler – its already the difference calculated by the reader for the other memcpy. So this is done in the reader and we save some bytes and have to do an subtraction in place for it.

## **id**

Next is the buffer id. Every buffer gets one and its an atomic counter so here we have a unique id. This makes things easy in checks for buffer use.

## **chksum**

Next a check sum. Its now not in use, but if you encounter problems and you assume a memory model problem you can make it and then try the fence for the transfer. This way I found my first two synchronization problems. If you need to check set the checksum flag and it is used.

## **safeguard**

The safeguard. Every structure that is potential a target for an memcpy should have one – you never can trust a corrupted log, and this even spawns multiple processes. So this is the least I can do for checking.

## **number\_dispatched**

We have for the thread a counter and so we can see when a buffer was made. This helps to identify the last buffer – which contains the best thread statistic counters.

## **counter\_write0 to counter\_write2\_adaptive\_very\_fast**

These are the static counters for the thread when the buffer is made. So you can get them here – its not possible to make a process wide counter without introducing an interference between the threads. So we have the thread specifics that could easy interfere in thread local storage and every time a buffer is made we get them here.

## **info**

The last we have here is the index to the real data for the log. Its again an int, so for the size I plan its big enough.

So this is the buffer control structure in the area – you will only need the info if you plan to do this by a new thing – or make the verify better. For now its used internal and for the tools, namely verify and defect.

Next is the area itself.

## The area

And here is is.

### ***shmversion***

We have first the version. So we can detect a mismatch for it. Helps a lot if you have different versions on your box.

### ***shmid***

Next is the id of the shared memory. This is used as an info, and as a check value.

While the version is something easy the shmid is normally a system given number and unique. So we can use that to check against the multiple init thing – simply set it in an int and if we do an init again we stop here.

### ***shmsafeguard***

Next a safeguard, as always.

### ***Shma shmf***

Now we have the two atomics that represent the anchors of the two lists for available and full buffers. Please note that I use here the index of the buffer, and so a 0 is NOT end of the list. Its a -1 this time

### ***shmcount***

Next we have the count of buffers from init. Note it can differ from the create – so you can play games with getting some more memory – but be sure to understand the full layout of the area or you will end up in memory overwrites in a shared memory buffer.

### ***ich\_habe\_fertig***

The next is the system wide log flag. See the source for its name and purpose.

### ***Readerflag readerpid***

Now we have two times an flag and a pid, one for readers and one for writers.

This is our little morse code approach to communicate with the area between programs. And for the readers its already in use.

### ***Writerflag writerpid***

For the writers is have no need for now, but its there and when I need it I have it here.

## ***logbuffers***

At last the descriptor array. The real buffers are after the descriptors. And yes, I use here a very dirty c trick.

See the init for how its done.

Now we come to the one and only buffer in the client

## **The tbuff struct in client**

This is the client and its local log buffer. Its used there and will not be used in the area. So the area is a different place. Keep this in mind.

First we have the three lists for the buffers.

### ***next\_cleanup***

One to connect them all. So this is the cleanup list.

We call it also the s list – sequential.

We only do this one time, but we still use an atomic for it. So we are save for now its not corrupt – if you don't overwrite it.

The cleanup at exit uses the list to check all buffers for log that is not already at the shared memory. So this is a vital one. Most simple programs don't need a flush and so they simply exit and leave the transfer of the log to this mechanism. This is also the reason we cannot use for the buffers thread local storage – would be nice, but simply does not the trick when we exit the thing.

We also use it in the reuse check with a threads tid now.

### ***next\_full***

The next list is the full list. We have a helper function to put a buffer on the list, and then its up to a slave thread to move it to shared memory. So this can be 0 in case we are not on the list – or we are the last buffer.

### ***next\_append***

Next is the list for available buffers – meaning we are not in use for a thread if we are on that list.

Don't miss that - the buffer normally is on the list first and when a thread need one its given to the thread then. Don't confuse this with an empty buffer – that's a different thing.

### ***safeguardfront***

Next is the usual safeguard.

### ***Pid tid***

Then we have again the pid and the tid.

## **acquiretime**

Now its time for a new time – or precise a clicktime difference. The acquire time is used to check for the time a buffer needs to be fetched from the available list . This can include the time to allocate it in case its a dynamic allocated one. So this is vital in getting info about the malloc or calloc overhead.

## ***id***

Next is the id. So we know that one and only buffer is it.

## **size**

Now we have the size. A buffer can have a size from 0 to the max. And this can be smaller than the max define if you reduce the max size.

## **maxsize**

So we have also the actual max in here.

Both are used to determine if the buffer is full – which does not mean that size and max is equal – we do not split log entry's, so we declare a buffer to be full if the actual payload is too much to fit in.

This has a big consequence – if you log big data its easy to fill buffers half or so. So be sure what you log.

## ***dispose***

Now we have a little but important flag. Its used to say goodbye for the thread to that buffer. If you finish logging for a thread this is set and then the slave does put it after the transfer to the area on the available list – so its a definitely goodbye. No way back.

## ***dispatched***

Next is the only atomic I need for the buffer. The dispatch flag. A full buffer becomes this set when it is moved on the full list. So the thread can now skip it if it check for a usable buffer. The slave resets the flag when the buffer content has been copied to shared memory. This means we use this also in the synchronization of the atomic to make the buffer for real in the slave thread – without synchronization we can easy end up corrupted. There is also some fence stuff here.

## ***number\_dispatched\_to\_counter\_write2\_adaptive\_very\_fast***

These are set with the thread statistic counters from thread local when we dispatch the buffer. So the thread local gets out to the reader this way and the converter can use it then.

## ***b***

Last is the pointer to the real buffer for the log – there you have the stuff you write .

## The thread locals

This is the helper we need in the thread to get things together from log to log. Its a struct so we use only one lookup for thread local at all.

The struct is also used as a kind of identifier – its address is unique. And the tid inside is used if you inspect it from outside.

*i*

First we have the pointer to a slave local which is for slave threads. For normal threads its 0.

### ***atrshmlog\_idnotok***

Next is the major flag to signal a vital logging. If you init a thread but have no vital logging its shut negative. If its 0 its OK. For the initial test its a -1. If you turn off the thread it is set to 1.

So this is a vital flag.

### ***atrshmlog\_targetbuffer\_arr***

Next are the pointers to the log buffers for the thread. We mean the in memory buffers here.

### ***atrshmlog\_targetbuffer\_index***

Next the index for the actual buffer.

### ***atrshmlog\_shm\_count***

Next the count in shared memory.

### ***strategy***

The next is the strategy flag . Again see the write for it. The basics: we can use different strategies for the threads to handle the all buffer full thing.

### ***Autoflush***

The next is the autoflush. Check again the write code for this. Its basically for the post mortem debugging here – you can set it and the content is moved to shared memory every write. You can set it to move via slave or direct. And yes, it slows down a lot.

### ***atrshmlog\_thread\_pid atrshmlog\_thread\_tid***

Then we have the pid and the tid.

### ***number\_dispatched\_to\_counter\_write2\_adaptive\_very\_fast***

Last the thread local statistics.

That's all for now.

## The slave local

And here it is.

### ***next***

We start with the link of the slave list. So we can iterate over the list by starting in tpslave and end up with 0. We have a function for that.

### ***tid***

We hold the tid. So we can do the iterate and find the slave and its tid in the C module notation. It can differ from the notation the system uses for threads (for example if we use pthread id and the system has a low level construct) or it can be different from your thread system id (for example if you use a own threading system and the tid of the system of yours is different from an OS tid).

But for starters – you should be able to make a match and then use it for the slave thread in question.

### ***g***

Last is the thread locals adress. So we can navigate to the thing and set its flag.

## The externs

Then follows a list of externs. These were in the former version static in the one and only module file. With the split of the module they are now needed extern. But they are after all internals, so don't use them in any code.

## The macros

So for now the header contains also some macro stuff.

Some helpers first.

The statistics counter setter is easy.

The rest are the layers for create, delete, cleanup and init of area.

Use of these is normally only in the programs of the module – exceptions are the other layers here.

So I decided to put them here and not into the interface header – perhaps I will change it, as far as I can see its for the four OK to be there now.

Last are the functions, but you know that already ...

## Real code

Back to real code. The real C code. The one and only.

Former versions were implemented in one file. So I had a library, but all binary stuff contained the whole thing.

That felt not right, and so I have split the thing into pieces. Most of it is now linked in if needed. So you find now most of it in the `impls` directory. This makes also patches easier.

To make the thing short : we discuss from now on the `atrshmlog.c` stuff, but then we only discuss the functions, not the files in `impls`. So you have to do a grep to find it in `impls`. But that's OK for me. The naming of the files should at least give a hint.

### ***atrshmlog.c***

Start with `atrshmlog.c` now.

We have the usual comment stuff here.

Then the include – only one needed.

Next is the thread local struct. We always need it in the code so its here the definition.

You can see we initialize it with the array to 0 and the flag to -1.

Next is the getter for its address.

If we are on a platform that does not support thread local storage we use the pthread specific approach.

We have a global key and a key init flag here.

When the attach or the getter tries to get the thing it checks for a valid key with the flag.

If its needed the key is initialized.

In the getter we use the value and make a dynamic allocated struct for the thread if it has no value for the key.

If all fails we return 0 so the caller has to handle it.

And this is all we have in here. The rest is up to the `impls`.

We will do them in a loose order of area related, buffer related and support stuff.

### ***atrshmlog\_attach***

We have to attach to a shared memory buffer.

We do this normally first in the programs. At least when we need a working area. Only the support

programs and the create and delete don't use it.

So this is the first stop for us.

And its a big one. In fact the biggest C source at all for now.

The thing is simple.

Use the environment variables or the flag files and get it up. Try to connect to the area. Then initialize the rest inside the program – NOT INITIALIZE THE AREA.

So if this code is done we have one of two things. A connect to the area or a null pointer and the flags set to 0. No logging. No events set. No values of interest.

The thing starts with some helpers. Perhaps I give the putenv a try in the next version for the interface, but for now its off – crashed in the SWIG perl layer .... Made a hack to make it run again, but for now I leave the switch in here.

For the systems without thread local storage we have the pthread specific stuff in place.

Same for the init. Its a bastard to cover the two worlds of variable and files.

Next we have a helper struct. The real thing is then inside the attach function.

The attach should initialize the module. So we prepare for the worst and use atomics here to make multiple calls impossible.

This can lead to a busy wait but that's OK for me – you start your program and you call then attach – so if you encounter multiple instances running you have at least a strange sense of doing a module init....

After it has locked the once atomic it checks for its own helper once flag. That's also used from some of the functions. So don't spoil it.

If we are on a platform that needs the pthread specific we do that init now – if its needed.

Next is the time thing for the initime.

Then to the core function: get the shared memory id – this is also the check for the need to use flag files. Its a bit a slow down if you start a program without logging, but its no big deal. If you don't want to make file stuff simply set an invalid shmid for it. 0 is perfect....

Having the shmid is then the next thing. Now it starts the connect . And that's the place where we have the first time the platforms.

Needs a helper on mingw to get the memory mapped file connect. See then that helper code if you need it. For the others its the usual shmat call.

The result determines if we are out – no area – or in. And this is the one and only time. No reconnect so so. I know, perhaps that a nice for some silly platform, but not for the rest of us.

After an mismatch we set the shmid to 0 and that's it.

For the rest we check at the end.

If its OK – we have a connected memory – things are as expected. Set the pointer and the id for the process.

We also set the env – seems a bid odd but for the flag file thing we better do it. So its a bid oversize for the env – its set after all – but OK for me.

Update: the putenv is for now only active if you set an define. The perl makes a core otherwise....

Now we use the helper array and set up all the things we want to know from the init things – variable or flag file does not matter, but we can use only one of them.

So then we use the suffixes, set the internal variables and that's it.

After the thing we initialize the buffers if this is needed – see the mingw for this.

Next we init the event lock buffer – if needed we make it bigger by using the dynamic memory.

The init is a bit bigger than you think, but its only a helper for the attach. For now its in a separate file, but I think I will change that. Its only needed here after all.

Next we set the atexit helper and this is vital for us. So the result for the attach is set to the success of this.

Then we start the slaves. Default is one, but it can be 0 too. This is to cover some simple programs that don't log much and end simple and clean in the exit.

After the slaves are up we check if we do logging from here on. If we don't its the program that can switch it on. If you have a tricky start up and want the thing initialized, but not running you start it off. Then you have the control and can switch on when its time.

The rest is to set the shmid.

And now we have a small trick to do here. We store the pid in this special variable. If we have a so called fork clone we can use this and the in place pid's for the old buffers to skip them – they are a relic of the former father.

Last things are the switch of the two initialized flags – the one for the others and our own.

After we are attached its possible to use the area. We have not initialized it – that's a separate thing here.

And we have not accessed it. Its only attached and we hope the best for the use of it.

Now we can check for the use of it. First we do the simple thing. Init it.

## ***atrshmlog\_init\_shm\_log***

We have to init the area.

So you have an area but its only raw memory. You need to init it.

Yes, we could have done it in the create too – but then we would have combined two very different

things – the creation and the use. So I have them still separate here.

Init uses a helper to do it for each buffer.

The thing is easy to read, but be careful for the parameters. We use the address of the area and then the index of the buffer. The rest is simple with the exception of the append.

We set it to the index plus one, so we point to the next buffer.

This is done with the design of the area in mind. So don't spoil it.

Back to the main function.

After the checks we have the fence 12 here. If you encounter problem on your platform for the access to the area set it on.

We check then for reinit – its forbidden.

Next we calculate the number of buffers ahead for the log buffers and use that as start for the log data buffers. There is some adjustment stuff to hit an aligned memory number – its a simple one and an alignment for 16 is good for all platforms I know.

Next we init the buffers with the helper.

When we have done it we correct a little mistake for the append list.

The rest is more or less simple. Only keep in mind we use a -1 as end of list here.

After setting the shmid the thing is official initialized.

At last we use a fence again here – if we need it for the access we also need it to announce our changes to the others.

## ***atrshmlog\_cleanup\_locks***

We cleanup all resources that we hold in the area.

This is needed in case we have OS resources here. This was the case with the mutex things, but now we only have atomics. To have the step is now an option. But if you plan to reinitialize the area this is a nice helper.

We change the states for the buffers. Then we switch the shmid and the safeguards.

## ***atrshmlog\_verify***

We have to check for consistency.

The verify has a fence to make sure you have it in your thread. Its off for now.

First we check the id in memory and in the area.

Next is the version.

It follows the check for every buffer.

For now we don't check the log buffer itself.

### ***atrshmlog\_create***

We need the buffer from the OS.

First we check the parameters, and here we have a special for mingw. It uses a naming internal and the parameter is not an OS thing, but an index in a names array. Its made to work for index 1 to 32 for now.

We calculate a full buffer size, which is the area struct and the bunch of buffer structs. We have here to use the trick from the common c community of having a dummy array in the area and adding the rest of the array size to it for the buffer structs. After the structs the log buffers follow last.

After that we set the flags and do the low level call for the posix systems and cygwin. For mingw we do the file mapper.

Result is a positive number in case of success. Its the so called shared memory id.

We need it as the ID in the environment for the attach.

### ***atrshmlog\_delete***

We need to destroy that buffer.

For the posix and cygwin we use the OS call. For the mingw its a dummy. That system uses reference counting. So we have to detach from it or the end of process will do the thing. The last process that disconnects then also destroys indirect by getting the reference count to 0.

So no code here for mingw.

### ***atrshmlog\_get\_area***

We get the area address.

Simple getter for the pointer.

### ***atrshmlog\_get\_area\_count***

We get the area count.

We check and get the area count here.

Note: It is rare used, so I didn't make a fence here, but if you need it sent me a post card and I do it.

### ***atrshmlog\_get\_area\_version***

We get the area version.

We check and get the area version here.

Note: It is rare used, so I didn't make a fence here, but if you need it sent me a post card and I do it.

### ***atrshmlog\_get\_area\_ich\_habe\_fertig***

We get the system wide logging flag.

We check the parameters and then get the flag.

Its here an atomic. So no fence.

### ***atrshmlog\_set\_area\_ich\_habe\_fertig***

We set the system wide flag for logging.

After we have checked it we get the old value.

Then we set the new.

We return the old value.

### ***atrshmlog\_transfer\_mem\_to\_shm***

We transfer a buffer to the area.

After a lot of checks we take a time to calculate the transfer time.

In case you encounter problems you can then make a check sum. It slows down a lot, but in case you do have problems its the only way to do it right to detect.

We then loop till it is done.

We get the top of the available list in the area index.

If no buffer is available we sleep and try again.

To be sure we don't hang for ever we check in that sleep loop two of the flags, the system wide and the final.

After we got a valid index we do the pop from stack thing. Check Williams on the memory orders if you have a question for them , in doubt you can contact me .

After we got a valid and made the pop we can now safely use the buffer.

We cut off this one from the available list.

Then we set the values.

The transfer of the log can be made in a loop instead of memcpy if you are hunting a synchronization problem.

Last is the time stamps for the calculation. The before time is also used with the starttime to calculate the time for the transfer.

The time is then set in the atomic variable for the process to be used in the adaptive wait strategy in write. See below.

Then we can use a fence 5 if we encounter problems of synchronization.

Last is the state switch for the buffer, so it can be found with our read functions later.

Then we push it on the full list for the area for the read fetch.

### ***atrshmlog\_read\_fetch***

We transfer a buffer with log from the area into process space and clear the buffer in area for reuse.

First we set the length of transferred data to 0 – this is needed to make a return without error but also without log valid. We then simply skip that try.

If we have not done it we initialize our thread local buffer.

We check the valid flag – if we are not valid we leave.

Next we check the area safeguard. This is a bit risky, we don't have an atomic here, but the thing normally never changes, so its OK for me.

Then we fetch from the full list.

If we get a valid buffer we can use thread fence 4 for a full sync.

We then check the sizes to detect corruption of it.

We have a bit spare memory in place, but in theory the size has to be positive and best equal to the max size.

A 0 is possible here.

In case of a 0 we set the state to free and push it again on the available list.

If the size is not 0 we take the start of transfer time timestamp.

Then we transfer the log.

Again we can do the thing in a copy loop in case we encounter synchronization problems or with a memcpy.

Now we can make a checksum and compare to the delivered one – which means both writer and reader must make the checksum. So we can detect fence problems.

Next are the copy of the values and the end of transfer time stamp.

Again we switch the state to free and push the buffer in the area to the available list.

### ***atrshmlog\_read***

We transfer a buffer from the area into process space , old version and no outdated.

This will be removed in a later version.

So in short its the read fetch, but not with the list stuff. We check for the state and that its.

The index is fix. So the check is done for one buffer and its up to the process to decide which it is.

Because we don't maintain the lists its only there to show the old days way.

### ***atrshmlog\_alloc***

We get a new buffer for the threads for logging.

First we check the available list for a buffer.

If we get one we return its address.

If the available list is empty we check if allocation of dynamic memory is available.

If not we leave with 0 pointer.

We lock the atomics flag and recheck.

After that we unlock and leave if no alloc is possible.

In case we can alloc we get prealloc count buffers with one low level alloc.

If the alloc fails we set the no more alloc flag and leave with 0.

We init the buffer descriptors and connect the new buffers to the available list.

We leave the lock.

If we got the raw memory we can use thread fence 8 to make the memory available in case we have synchronization problems.

We try again to get a buffer from the available list .

This is a pathologic case here, in principal we could have a lot of threads waiting, and after we got the work still end up in eaten up all available buffers before we can pop from the available list. So its mandatory to start again with the initial check, even if its normally clear we have now buffers on the list.

After we got a valid buffer we leave.

### ***atrshmlog\_il\_connect\_buffers\_list***

We initialize the buffer descriptors and connect the buffers to the lists.

We initialize the buffer structs and connect them for the append and cleanup list pointers.

Then we add them for the last buffer the anchors of the cleanup list and push the top on the stack - this is not a simple push, but a push of the whole bunch of buffers in one atomic operation.

Same thing then here for the append list.

Note that the thing can also do the connect to the log buffers if the parameter is set.

If a buffer is already initialized it can be simply pushed on the lists with the thing – no initialize then with the log buffer, no loop, only the two push's.

### ***atrshmlog\_acquire\_buffer***

We get a buffer from the alloc and initialize it for the thread.

First we take the time to make the acquire time in case we succeed.

Then we call atrshmlog\_alloc to do the low level stuff.

After we got a buffer from alloc we set it initial.

Last thing is to take the end time and calculate the acquiretime.

In case of problems we leave with a 0.

### ***atrshmlog\_dispatch\_buffer***

We push a buffer from a thread on the full list.

We check the buffer is for real and we have it not yet on the full list – this is done with the atomic dispatched flag.

If it is dispatched we leave.

Then we use in case of synchronization problems fence 1 to get the buffer in.

In case we need checksums we calc it here.

We set the dispatched flag now and then push the buffer on the full list.

## ***atrshmlog\_free***

We give up ownership of a buffer.

The flags are set to reused and then the buffer is pushed on the available list.

No cleanup or transfer is done.

This a simple helper.

## ***atrshmlog\_flush***

We push our buffers to the full list.

We loop and push them with dispatch on the full list.

This is done with a non zero size buffer .

In case the dispose flag was set the buffers are then no longer available for the thread. So this has to be done also even with an size 0 buffer.

After the call you have to wait for logging at least till the buffers made it to a slave proc and have been cleared for the dispatched flag. Or you have still non dispose and empty buffers in place.

## ***atrshmlog\_write0, atrshmlog\_write1, atrshmlog\_write2***

We write to the log.

We have three write functions. Beside the handling of payload they are equal. So this is for all three write functions.

We get the thread local buffer.

If the thread local is not initialized we do that.

So a thread only use to log – no other thing needed. Init is done inside the first log.

We check the parameters.

We then check the eventflag for a point in time logging.

In this case we check the starttime parameter. If its 0 we get a new starttime. Then we set the endtime with the starttime.

If we have not a point in time log we check the endtime parameter. If it is 0 we get an endtime.

This is the hidden mechanism to make less calls in the layers for other languages. In C and C++ it is no big advantage against the taking of times in advance.

Next we do the argv array concatenation in case of write 2.

We then calculate the payload length.

We start the buffer checks.

We have to check for an not dispatched buffer first.

If we have no luck for our buffers we use the thread strategy to decide what to do.

We can discard the log. So we loose it but we have no penalty for a wait time. This can be used for threads that are critical to do the job fast but still should give info .

We can wait for the wait time flag value. This is the default and needs to work good a working nanosleep function at least. Doin't know if we have luck on f.....

We can spin loop. This is CPU consuming but fast if we have only seldom a hit for it.

We can wait adaptive. This is done by using the last transfer time for a buffer and make a simple assumption about the average wait time. We use the count of buffers to break it down and then try to sleep that time.

So for a fast memory transfer the time is short, for a slow it is long. And its in regard of the last done time, so we have a kind of self adjustment here.

To work faster we have a version with a slash down to half and tenth the time.

You have to check for the times you have for the buffer transfers to decide if the thing is worth the use – in case of a relative stable time a simple wait with the proper time is better.

We check in the wait case also the final and logging flags every iteration, so we can be stopped by those flags.

After we got an not dispatched buffer we can synchronize with fence 2 if we encounter problems.

Then the buffer's max length is tested against the total size – it is possible to be exceeded in the case a reduction was done for the size and we have a dynamic allocated buffer at hand.

Next we check if we have the needed space free. This is done against the actual buffer sizes.

If the buffer has not the amount free we dispatch it and start over again with the dispatch loop.

This can mean you have with a big logging in short time all buffers dispatched and then to wait. For small payloads its rare to hit the size limit, rough 10000 logging's are needed to fill a buffer with a small payload.

If the buffer has sufficient memory left we calculate the actual position in the log buffer and transfer the parameters. This is done by memcpy for now, later I try to switch to a better – faster – way.

In case of write 1 and write 2 the payload and also the argv concatenated buffer is appended. The argv is limited to 64 K in this version, you can make that bigger but there is a buffer in place so don't overdue it.

In the layers the argv for write2 is not implemented. You have to concatenate the arguments there by yourself.

After the transfers the size is updated.

If we have the autoflush off we are through.

If autoflush is not zero we prepare the transfer to shm.

Then we dispatch the bufer to the slaves if the autoflush is set to 1.

If autoflush is set to 2 we do the checksum if needed and directly move the buffer to shared memory.

In case of the write 2 we have then the concatenation of the argv array – its the classic C argv approach. We use a helper buffer here. Later I will switch back to a direct copy, its faster and we don't have the helper buffer any more. Still the thing will be limited to not hurt the size limits to early.

### ***atrshmlog\_init\_thread\_local***

We initialize the thread local struct.

We check for already init.

If its init we deliver the flag .

If it is not init we clear the buffer pointers, then we check for a valid connect area.

In case of non connect or version problems we leave with flag set to no logging.

In case the area is OK we set pid , the tid , the strategy from processand autoflush from process and the thread statistics.

### ***atrshmlog\_init\_in\_write***

We init in the write function.

We first init the thread local buffer.

We then lock an atomic lock flag.

We then check for a fork clone situation. This can happen in case your program forks but does no exec. A shell is an example for this thing to happen.

You then have to start the slaves for the forked process or you will not log till end of process happens with a cleanup.

So the memory contains in a fork clone the old pid of the father. We check that against our pid. If the pid is a mismatch we have a fork clone process. So we start the slaves and then reset the process pid.

After the fork clone check we clear the flag – if another thread was in, it will then only see the already new pid.

Then we get the buffers.

If we encounter a problem we give free the already allocated buffers and then leave nonlogging.

### ***atrshmlog\_stop***

We stop logging for this thread.

We switch the logging flag in the thread locals to off.

Then dispatch the buffers with a set dispose flag. So they will be moved to the available list after the transfer to the area.

To be sure we don't use the buffer any more we set our pointer to 0.

### ***atrshmlog\_turn\_logging\_off***

We stop logging for possibly another thread.

We get the logging flag for the thread determined by the thread local struct.

We then can use fence 13 in case we have synchronization problems.

We switch logging off and we dispatch the buffers with the dispose flag set.

So the thread cannot log any more and its buffers are on the full list and will be pushed on the available list.

The thread can be a slave thread. Then there are simply no buffers. The flag is switched and the thread will end itself when it loops in the next iteration. So this is a graceful stop for the slave thread.

WARNING:

This will likely CRASH if the thread has ended – there is simply no more thread local then.

So you cannot use it after a thread has ended or you killed it.

### ***atrshmlog\_reuse\_thread\_buffers***

We have to reclaime dead buffers for killed threads or threads that ended and did not stop logging.

We get the head of the cleanup list.

Then we iterate the list.

For every buffer we check first the atomic, then we can use fence 10 if we encounter synchronization problems.

We skip if the safeguard is corrupt.

We skip if we have the wrong tid or pid – eventually a fork clone father in place.

We set the dispose and then dispatch the buffer, so its content is back to the area after.

We deliver at last the number of buffers we found.

This should never be used for a still living and logging thread. Use only if you kill threads – any other thing makes a cleanup.

If you end threads without stop in the thread you should do this to reclaim the buffers after you are sure the thread is gone.

WARNING: Don't forget that the tid is the Module tid here.

### ***atrshmlog\_exit\_cleanup***

We cleanup the buffers at end of program.

First we check if logging is on with a valid area.

Then we initialize our thread locals if needed.

We set the process wide logging to off.

We set the slave flag to exit to on, so slaves will in the next iteration leave the loop and exit.

If the wait for slaves is active we then loop till we find all slaves are down. This is off by default. It works only if the slave cont is accurate, so if you kill slaves you have to use the decrement slave count to balance the slave count, too.

We get the actual pid to detect a fork clone scenario.

We cut off the cleanup list and then iterate it.

We use fence 10 if we have synchronization problems.

The buffer is dispatched if it contains log and the fork clone test detects no father buffer. So only buffers of the actual process are transferred to the area.

Last we switch the final flag on so no more logging and slave activity takes place. This cannot be set back so its final.

### ***atrshmlog\_create\_slave***

We create a new slave thread.

We get a piece of memory for the slave local.

We use the thread start function of the OS and start the new thread.

We deliver the result as a return value.

We deliver in case of c11 and pthread the thread id in the global variable `atrshmlog_f_list_buffer_slave`.

This is not good enough, but for now its the thing you get. Will change this in a later version.

### ***atrshmlog\_f\_list\_buffer\_slave\_proc***

We iterate the full list and transfer log buffers to the area.

First we init the thread locals buffer if needed.

Then we put our slave local on the slave list. So we are now part with our slave local on that list.

The slave local is made in the create, and we use it here – so we are responsible for setting it.

We maintain the slave counter.

We then enter the iteration loop. Its controlled by the flag

`atrshmlog_f_list_buffer_slave_run`, so you can stop all threads for the next iteration by it.

In the loop we check the thread local flag, if this is switched we leave.

We check for an buffer on the full list.

If we don't get one we sleep. In the check loop we leave if the final flag is set.

If we have a buffer we check the dispatched flag.

Then we synchronize if needed with fence 6 the buffer.

We check the size and transfer it if it is not 0 to the area.

After the transfer we synchronize if needed with fence 7.

We clear the dispatched flag. If the dispose is set we free the buffer.

So the thread that holds the buffer is responsible to no longer use it in that case.

After that the iteration starts again.

When we leave we maintain the slave list and the slave counter.

### ***atrshmlog\_decrement\_slave\_count***

We decrement the slave count.

We decrement till we hit a 0.

This is needed if you kill slaves. Then its up to you to maintain also the slave count and the slave list. You have to use this in this case. Don't use it elsewhere.

### ***atrshmlog\_remove\_slave\_via\_local***

We remove a slave from the slave list.

We need this if we kill slave threads to keep the slave list intact. The list is made for the slave locals. So you have to use this after you kill the thread.

First we make some checks and then we iterate the slave list till we hit a 0, a self or the former in the list.

If its a 0 the slave is already off and we leave.

If its a self we are the top of the list and we pop from it the pop then. No further check and we leave.

If we hit the former on the list we use our next member to set the former and so leave the list.

### ***atrshmlog\_get\_next\_slave\_local***

We get the next slave local from the slave list.

If we call it with 0 it will get the top of the list.

Else it will deliver the next for the given slave local.

This works only if the list is intact. So don't use it parallel with turn off or with remove form list.

### ***atrshmlog\_turn\_slave\_off***

We turn a slave off.

We set the ok flag to off and the rest is then done when the thread again hits the flag in the iteration loop.

So this is no kill, but it should stop the slave withing the next second.

It is the prefered way to stop a slave, do not use it together with a kill of threads.

If you kill you have to maintain the slave list and count by yourself.

### ***atrshmlog\_get\_env***

We get an environment variable value with prefix and the suffix.

We build the full name and then get the value.

### ***atrshmlog\_get\_env\_shmid***

We get the shared memory ID from environment.

### ***atrshmlog\_get\_env\_id\_suffix***

We get the ENV SUFFIX value.

### ***atrshmlog\_get\_env\_prefix***

We get the prefix.

If we are executed first time we use the default prefix to check if another is given as environment variable. So you can redefine the prefix by setting a variable with its name.

After the check for all further calls the prefix is fix.

### ***atrshmlog\_set\_env\_prefix***

We set the prefix.

We check if the prefix buffer is already set. If yes we leave.

Else we move the parameter text to the buffer.

This is not mt save, but I think its OK for me. Simply do this only at one place and best with attach after it.

### ***atrshmlog\_buffers\_preallocated***

This is not only code, more about the static buffers.

We use static buffers to speed up the logging for the used buffers. This is done by using the buffers as static array with connected entry's. Also the log buffers itself are done static.

The thing is initialized with the Macro

ATRSHMLOGBUFFERS\_PRE

for the first buffers.

The last buffer is initialized with

ATRSHMLOGBUFFERS\_PRE\_LAST

Its mandatory that then buffers are initialized in the given order and with the correct index number.

The last must be also the ATRSHMLOGBUFFER\_PREALLOCATED\_COUNT – 1 index.

Anything else and you will encounter buffer shortage at best, crash landing at program start or worse unpredictable program log run.

We use for now 64 buffers. That's enough for 32 threads for use of two per thread.

If you plan to use the log in a high number of threads environment you should raise the number to match at least your starting values.

If the static buffers are fetched by alloc from the available list an acquiretime withing 200 clicks is OK.

If you encounter much higher costs check your platform for problems with static variables and initialization.

Keep in mind that a big number of buffers always mean you need more memory. So this can cost time in the program. But for most platforms a static buffer has no run time cost. So this is preferable over the dynamic allocation.

There is one helper to init the chunks for the log buffers. This is switched on in cygwin and mingw. So the log buffers are not only static but also accessed. This cost some time at program start up. But it reduces access time for logging drastically in the write functions.

On Linux the helper is not needed, but you can switch it on if you need.

### ***atrshmlog\_il\_get\_raw\_buffers***

We get new buffers from the dynamic memory.

The buffers are fetched with one low level alloc for the number of buffers and the size for the log buffer itself.

The total size is allocated for one low level call, its a malloc for normal and a calloc for init in advance case.

If there are new buffers the alloc count is maintained.

### ***atrshmlog\_get\_logging***

We get the logging state.

The first is to check for a connected area. If not connected we don't log and leave.

There is a fence 10 in case we encounter synchronization problems.

We check the system wide log flag in the area and leave if we are not logging.

We check the final flag and leave if we are not logging.

We check the process wide flag and leave if we are not logging.

Else we log and return that.

### ***atrshmlog\_get\_realtime***

We get a real time.

We use the best approximation for the platform we have for now.

We set the seconds and nanoseconds part then and return.

In case of mingw its the lifetime with a precision of 100 nanos. In case of the posix and cygwin its the `clock_gettime`.

If we don't have them we try `gettimeofday` and calculate the nanos from the microseconds.

### ***atrshmlog\_get\_statistics***

We get the statistics counter array.

First we deliver the actual click time in low and high part for the field 0 and 1.

Then we deliver the statistics counters.

The array must be big enough or you overwrite memory. To make it right you use the `atrshmlog_get_statistics_max_index` and add at least one and then make the buffer.

For now there are 86 counters, so an array of 100 is doing fine.

### ***atrshmlog\_sleep\_nanos***

We sleep nanoseconds.

For the platforms with nano sleep its doing a loop for every 100000.

For the platform without it calculates the milliseconds and does that.

At least it calls a sleep 0 for it.

So you have to check your platform. In case of fenster;plural it simply cannot do better than milli. Don't try to use a busy wait here. This is not what you want in the end for the module.

### ***atrshmlog\_set\_event\_locks\_max***

We set the event locks limit and adjust the buffer.

The new limit is checked and if its smaller nothing happens, only the limit is reduced.

If its bigger a new buffer is allocated from malloc and the old part is copied in. The rest is initialized with 0.

### ***atrshmlog\_init\_events***

We initialize the events in the event buffer.

We use environment variables to do it in the first way, files in the second.

First way.

We get EVENT NULL and EVENTONOFF.

Then we decide with EVENT NULL to do positive or to do negative logic.

Positive is to set the buffer with 1, negative to set it with 0.

In positive then we use onoff to switch the given events off. In negative we switch them on. If a number is out of bounds for the event lock array its ignored.

Second way.

We do the same but use the flag file for NULL and a file with event numbers in onoff.

### ***atrshmlog\_get\_acquire\_count***

Simple getter.

### ***atrshmlog\_get\_autoflush\_process***

Simple getter.

### ***atrshmlog\_get\_buffer\_id***

Simple getter.

### ***atrshmlog\_get\_buffer\_max\_size***

Simple getter.

### ***atrshmlog\_get\_buffer\_size***

Simple getter.

### ***atrshmlog\_get\_clock\_id***

Simple getter.

### ***atrshmlog\_get\_checksum***

Simple getter.

### ***atrshmlog\_get\_env\_id\_suffix***

Simple getter.

***atrshmlog\_get\_event***

Simple getter.

***atrshmlog\_get\_event\_locks\_max***

Simple getter.

***atrshmlog\_get\_thread\_fence\_1***

Simple getter.

***atrshmlog\_get\_thread\_fence\_2***

Simple getter.

***atrshmlog\_get\_thread\_fence\_3***

Simple getter.

***atrshmlog\_get\_thread\_fence\_4***

Simple getter.

***atrshmlog\_get\_thread\_fence\_5***

Simple getter.

***atrshmlog\_get\_thread\_fence\_6***

Simple getter.

***atrshmlog\_get\_thread\_fence\_7***

Simple getter.

***atrshmlog\_get\_thread\_fence\_8***

Simple getter.

***atrshmlog\_get\_thread\_fence\_9***

Simple getter.

***atrshmlog\_get\_thread\_fence\_10***

Simple getter.

***atrshmlog\_get\_thread\_fence\_11***

Simple getter.

***atrshmlog\_get\_thread\_fence\_12***

Simple getter.

***atrshmlog\_get\_thread\_fence\_13***

Simple getter.

***atrshmlog\_get\_f\_list\_buffer\_slave\_count***

Simple getter.

***atrshmlog\_get\_init\_buffers\_in\_advance***

Simple getter.

***atrshmlog\_get\_inittime***

Simple getter.

***atrshmlog\_get\_inittime\_tsc\_after***

Simple getter.

***atrshmlog\_get\_inittime\_tsc\_before***

Simple getter.

***atrshmlog\_get\_minor\_version***

Simple getter.

***atrshmlog\_get\_patch\_version***

Simple getter.

***atrshmlog\_get\_prealloc\_buffer\_count***

Simple getter.

***atrshmlog\_get\_shmid***

Simple getter.

***atrshmlog\_get\_f\_list\_buffer\_slave\_wait***

Simple getter.

***atrshmlog\_get\_statistics\_max\_index***

Simple getter.

***atrshmlog\_get\_strategy***

Simple getter.

***atrshmlog\_get\_strategy\_process***

Simple getter.

***atrshmlog\_get\_tid***

Simple getter.

***atrshmlog\_get\_thread\_local\_tid***

Simple getter.

***atrshmlog\_get\_version***

Simple getter.

***atrshmlog\_get\_wait\_for\_slaves***

Simple getter.

***atrshmlog\_set\_init\_buffers\_in\_advance\_off***

Simple setter.

***atrshmlog\_set\_init\_buffers\_in\_advance\_on***

Simple setter.

***atrshmlog\_set\_buffer\_size***

Simple setter.

***atrshmlog\_set\_autoflush***

Simple setter.

***atrshmlog\_set\_autoflush\_process***

Simple setter.

***atrshmlog\_set\_checksum***

Simple setter.

***atrshmlog\_set\_clock\_id***

Simple setter.

***atrshmlog\_set\_event***

Simple setter.

***atrshmlog\_set\_thread\_fence\_1***

Simple setter.

***atrshmlog\_set\_thread\_fence\_2***

Simple setter.

***atrshmlog\_set\_thread\_fence\_3***

Simple setter.

***atrshmlog\_set\_thread\_fence\_4***

Simple setter.

***atrshmlog\_set\_thread\_fence\_5***

Simple setter.

***atrshmlog\_set\_thread\_fence\_6***

Simple setter.

***atrshmlog\_set\_thread\_fence\_7***

Simple setter.

***atrshmlog\_set\_thread\_fence\_8***

Simple setter.

***atrshmlog\_set\_thread\_fence\_9***

Simple setter.

***atrshmlog\_set\_thread\_fence\_10***

Simple setter.

***atrshmlog\_set\_thread\_fence\_11***

Simple setter.

***atrshmlog\_set\_thread\_fence\_12***

Simple setter.

***atrshmlog\_set\_thread\_fence\_13***

Simple setter.

***atrshmlog\_set\_logging\_process\_off\_final***

Simple setter.

***atrshmlog\_set\_f\_list\_buffer\_slave\_count***

Simple setter.

***atrshmlog\_set\_logging\_process\_off***

Simple setter.

### ***atrshmlog\_set\_logging\_process\_on***

Simple setter.

### ***atrshmlog\_set\_prealloc\_buffer\_count***

Simple setter.

### ***atrshmlog\_set\_f\_list\_buffer\_slave\_wait***

Simple setter.

### ***atrshmlog\_set\_strategy***

Simple setter.

### ***atrshmlog\_set\_strategy\_process***

Simple setter.

### ***atrshmlog\_set\_thread\_fence***

Simple setter.

### ***atrshmlog\_set\_wait\_for\_slaves\_off***

Simple setter.

### ***atrshmlog\_set\_wait\_for\_slaves\_on***

Simple setter.

### ***atrshmlog\_set\_f\_list\_buffer\_slave\_run\_off***

Simple setter.

### ***atrshmlog\_init\_via\_env***

Helper for init.

### ***atrshmlog\_init\_via\_file***

Helper for init.

## ***NON INLINE CODE***

We need for the inline functions in case we switch inline off implementations.

So for every platform we have them here.

We use the defines to make the one and only visible for the compiler we need.

In the inline functions we do for now only two things.

Get the click time.

Get the clock time.

Get the clock time is for now a simple switcher for the clicktime you need. Default is to use the simple clicktime.

For the platforms on Intel architecture we have possible a clicktime with a fence to hinder optimizations to move it around.

And there is an advanced clicktime with the fence thing already in place internal.

The rest are the click time getter's. They use inline and assembler. They are for the Intel platform for now.

See this one:

```
/**\n *\n * \n Main code:\n *\n * The simple version, no check for fences\n\nuint64_t atrshmlog_get_tsc_x86_64_gnu(void)\n{\n    uint32_t hi, lo;\n\n    __asm volatile\n        ("rdtsc" : "=a" (lo), "=d" (hi));\n\n    return ((uint64_t)hi << 32) | lo;\n}
```

We have here the gcc inline assembler in place.

To make it you need the gcc compiler. Another compiler and you have to make it there, or use a mixed approach for linker compatible objects.

```
$ nm atrshmlogimpl_non_inline_code_linux_gcc.o\n          U atrshmlog_clock_id\n0000000000000040 T atrshmlog_get_clicktime\n0000000000000010 T atrshmlog_get_tsc_fence_x86_64_gnu
```

```
000000000000000030 T atrshmlog_get_tsc_null_x86_64_gnu
00000000000000000000 T atrshmlog_get_tsc_par_x86_64_gnu
00000000000000000020 T atrshmlog_get_tsc_x86_64_gnu
U_GLOBAL_OFFSET_TABLE_
```

as you can see there is not much inside the object code – so if you can't do it for your compiler you can use the gcc in place to make the thing and then link the non inline to the library.

So it is possible to use them even if the main compiler does not support inline assembler.

Last resort is to make a dummy function, make the assembler code with the magic compiler switch -S and patch the file. Then run the assembler to make the object.

For a simple dummy we get

```
.file "ass.c"

.text

.globl atrshmlog_get_tsc_x86_64_gnu
.type atrshmlog_get_tsc_x86_64_gnu, @function

atrshmlog_get_tsc_x86_64_gnu:

.LFB2:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
movl $4711, -4(%rbp)
movl $4812, -8(%rbp)
movl -4(%rbp), %eax
salq $32, %rax
movq %rax, %rdx
movl -8(%rbp), %eax
orq %rdx, %rax
popq %rbp
```

```

.cfi_def_cfa 7, 8
ret
.cfi_endproc

.LFE2:
.size atrshmlog_get_tsc_x86_64_gnu, .-atrshmlog_get_tsc_x86_64_gnu
.ident "GCC: (GNU) 5.3.1 20160406 (Red Hat 5.3.1-6)"
.section .note.GNU-stack,"",@progbits

```

All left is to replace the

```

movl $4711, -4(%rbp)
movl $4812, -8(%rbp)

```

with the

```

rdtsc
movl %eax, -4(%rbp)
movl %edx, -8(%rbp)

```

after a quick compare.

Then assemble the thing and you are there.... At least if your assembler is the build chain part of the compiler.

That's a thing I have not to do, my compiler supports so far the inline assembler in place.

### ***All files with \_flag.c***

They hold the flag variables.

### ***All files with \_buffer.c***

They hold a buffer.

### ***All remaining files with \_list.c***

They hold a list anchor.

### ***Best behavior***

For most of the functions its easy. You use it only if you have to.

If you try something strange – lets say a atrshmlog\_get\_area\_count(13); - you are on your own.

If you try to do something you really need and you are in trouble – let's say you DO have to kill slaves – then this is another story. I have made the functions so far that you can do the thing, but I am best against this. So you can do it. But if you miss it for one of the things that you have to encounter simply sent my your full code – and I mean here full, not the two lines you have with the problem call in place – and I am in. Any thing I can not see for my self and you are off...

So use the simple rule of best behavior: Leave the place always a bit more cleaned up and nice then you have found it.

# **Appendix**

Here we have the usual lists.

## Illustration Index

Illustration 1: The BASEDIR after unpacking the tar ball.....	34
Illustration 2: The bin directory with the build scripts.....	35
Illustration 3: The clean src directory.....	36
Illustration 4: Check for complete script start.....	37
Illustration 5: Result for the check complete script.....	38
Illustration 6: Analyze the system with the check system script.....	39
Illustration 7: Setting the right environment.....	40
Illustration 8: After the makeall.sh finished.....	41
Illustration 9: Checking the build result with a ls.....	42
Illustration 10: A first test. We create a buffer for 8 log buffers with key 4711.....	43
Illustration 11: Initialize the area.....	44
Illustration 12: Testing the log with the hello world demo program.....	45
Illustration 13: Setting the area to no more logging and reading the data.....	47
Illustration 14: The content of the directory tree d1 after transfer.....	48
Illustration 15: Convert binary to human readable text.....	49
Illustration 16: The resulting log in text form.....	50
Illustration 17: Supported platforms in the atrshmlog.h header (previous version ...).....	51
Illustration 18: Detecting the OS and the architecture with uname.....	53
Illustration 19: Platform check for another system.....	54
Illustration 20: Check for a conform compiler and the atomic header.....	55
Illustration 21: Check for a C 11 feature, the _Thread_local.....	57
Illustration 22: Limits for the user.....	60
Illustration 23: A hello world program C source code.....	66
Illustration 24: A hello worlds internal stuff.....	67
Illustration 25: The include added.....	69
Illustration 26: Attaching to the area.....	71
Illustration 27: Adding the logging ( and correcting a nasty error too).....	73
Illustration 28: The build and a first test.....	74
Illustration 29: The output after a test against an active area (and some upses).....	74
Illustration 30: The deep stuff.....	75
Illustration 31: The online documentation for ATRSHMLOG_WRITE.....	76
Illustration 32: Hello world with use of argv and timing the printf.....	80
Illustration 33: The test for hello world with argv use and printf timing.....	81
Illustration 34: The java ase directory.....	90
Illustration 35: The java bin with its scripts.....	91
Illustration 36: The vendor's directories.....	93
Illustration 37: Transfer from BASEDIR/src to the vendor's directories.....	94
Illustration 38: Inside a vendor dierctory.....	95
Illustration 39: .. and more inside of it.....	96
Illustration 40: Inside the package.....	98
Illustration 41: Setting up the build environment.....	100
Illustration 42: Create the jni library.....	101
Illustration 43: Test of the jni bridge.....	102
Illustration 44: The python basedir.....	113
Illustration 45: The python bin directory with the scripts.....	114
Illustration 46: The python source directory.....	115
Illustration 47: Transfer of lib and headers before build.....	116
Illustration 48: Inside the source directory ready for build.....	117

Illustration 49: Setting the build environment.....	119
Illustration 50: Create the python library.....	120
Illustration 51: Test of the python library.....	121
Illustration 52: The perl basedir.....	129
Illustration 53: The bin directory with the scripts.....	130
Illustration 54: The perl source directory.....	131
Illustration 55: Transfer of library and headers.....	133
Illustration 56: Inside the source ready for build.....	134
Illustration 57: Setting the environment for build.....	136
Illustration 58: Build the perl library.....	137
Illustration 59: Test of the perl library.....	138
Illustration 60: Test log output.....	139
Illustration 61: The SWIG base directory.....	146
Illustration 62: The bin directory with the scripts.....	147
Illustration 63: The source directory for SWIG.....	148
Illustration 64: Transfer of the library and the headers.....	150
Illustration 65: Inside the source ready for build.....	151
Illustration 66: Setting the build environment.....	153
Illustration 67: Create the tcl library with SWIG.....	154
Illustration 68: Test the new tcl library.....	155
Illustration 69: The log result.....	156
Illustration 70: The cygwin base directory after unpack.....	182
Illustration 71: Copy for the adapted headers.....	183
Illustration 72: Compile the module with makeall.sh.....	184
Illustration 73: Time for a compile in cygwin on my box.....	185
Illustration 74: Getting an administrator's shell.....	186
Illustration 75: Starting the cygrunsrv for the service cygserver.....	187
Illustration 76: Create the shared memory buffer.....	188
Illustration 77: And initialize the area.....	189
Illustration 78: Running the first test.....	190
Illustration 79: Starting the reader to transfer the log.....	191
Illustration 80: Stopping the reader after its done.....	192
Illustration 81: Convert from binary to human readable text.....	193
Illustration 82: The result log.....	194
Illustration 83: The mingw base directory in a cygwin system and setting the environment.....	196
Illustration 84: Copy of the already adapted headers.....	197
Illustration 85: Starting the build with makeall.sh.....	198
Illustration 86: A crash landing for test03 .....	199
Illustration 87: Setting the path for a cmd.....	200
Illustration 88: Create of a shared mapped memory via pagefile.sys.....	201
Illustration 89: Initialize or the area from a cmd.....	203
Illustration 90: Running a first test from a cmd.....	204
Illustration 91: Running the reader from a cmd.....	205
Illustration 92: Using signal reader to stop the reader - from a cmd.....	206
Illustration 93: Using the convert from a cmd to get human readable text.....	207
Illustration 94: The result log.....	208
Illustration 95: Generation of the jni bridge with mingw for vanilla java.....	209
Illustration 96: Build of the jni bridge for mingw.....	210
Illustration 97: Test of the jni bridge with vanilla java and cmd.....	211
Illustration 98: The resulting log.....	212



## Error codes

Technically the error codes are made with int. So we use enum's, but at the interface we get an int back.

This has the advantage that we don't have any problems to do this with the other languages, and even the applications in C and C++ don't have problems with it. It simply works.

When it comes to use of higher level constructs an enum is still preferable. So we have also the enum in place. Later in development I switched to use the enum's too.

In practice we have a no error code. Its 0 – an old UNIX and C tradition.

Then we have for every function a range of ten or 20 numbers. All make a stop at the next full ten.

Positive is a minor error, sometimes simply as an all was OK but simply no data there thing.

Negative is serious. So this can only mean something went very wrong.

Best seems to use the raw int and first check for 0, then start any enum uses.

The enum is atrshmlog\_error.

Now for the errors a short list. We give the enum, its value, its meaning and a rational with possible reasons. Can be more than one.

### **atrshmlog\_error\_ok**

Value 0

The Operation was successful, no error. You can simply go on.

### **atrshmlog\_error\_error**

Value -1

A generic error code. Will be replaced in the next version with specific.

### **atrshmlog\_error\_error2**

Value -2

A generic error code. Will be replaced in the next version with specific.

## **atrshmlog\_error\_error3**

Value -3

A generic error code. Will be replaced in the next version with specific.

## **atrshmlog\_error\_error4**

Value -4

A generic error code. Will be replaced in the next version with specific.

## **atrshmlog\_error\_error5**

Value -5

A generic error code. Will be replaced in the next version with specific.

## **atrshmlog\_error\_connect\_1**

Value -11

Buffer list is NULL in atrshmlog\_il\_connect\_buffers\_list.

Rational:

Parameter error, check your code.

## **atrshmlog\_error\_connect\_2**

Value -12

Buffer count is negative in atrshmlog\_il\_connect\_buffers\_list.

Rational:

Parameter error, check your code.

## **atrshmlog\_error\_init\_thread\_local\_1**

Value -21

The init was not successful in atrshmlog\_init\_thread\_local.

Rational:

The init found a problem with the shared memory area.

## **atrshmlog\_error\_mem\_to\_shm\_1**

Value -31

The shared memory is not connected in atrshmlog\_transfer\_mem\_to\_shm.

Rational:

No valid connect was made – did you forget to attach ?

OR: Possible is an overwrite inside the process that nulled the pointer variable.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_mem\_to\_shm\_2**

Value -32

The transfer buffer is NULL in atrshmlog\_transfer\_mem\_to\_shm.

Rational:

Parameter error. Check your program code.

## **atrshmlog\_error\_mem\_to\_shm\_3**

Value 31

The size is 0, no operation in atrshmlog\_transfer\_mem\_to\_shm.

Rational:

Your buffer contains no log info. This can happen when you flush buffers or stop logging for the thread.

## **atrshmlog\_error\_mem\_to\_shm\_4**

Value -33,

The maximum size is exceeded in atrshmlog\_transfer\_mem\_to\_shm for the buffer itself.

Rational:

Your size is bigger than the maxsize for the buffer. You have an corrupted buffer in your program. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_mem\_to\_shm\_5**

Value -34,

The logging was off for the thread in atrshmlog\_transfer\_mem\_to\_shm.

Rational:

The thread does not log. The init was not successful or someone has switched logging off after.

## **atrshmlog\_error\_mem\_to\_shm\_6**

Value -35,

The area safeguard was corrupt in atrshmlog\_transfer\_mem\_to\_shm.

Rational:

You have a memory overwrite for the area. Stop logging and shut down that area. At least reinit it.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

## **atrshmlog\_error\_mem\_to\_shm\_7**

Value 32,

The logging was off for the system in area for atrshmlog\_transfer\_mem\_to\_shm.

Rational:

The system wide logging flag has been switched to off.

## **atrshmlog\_error\_mem\_to\_shm\_8**

Value 33,

The logging was final off in atrshmlog\_transfer\_mem\_to\_shm.

Rational:

The program shuts down so no more logging is allowed.

OR: Someone has overwritten the final flag, shut down the program. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_attach\_1**

Value 41,

The attach was already done in atrshmlog\_attach.

Rational:

The attach found a concurrent operation already made it.

Check your program code. Only one attach should be made because we do not support multiple area.

## **atrshmlog\_error\_attach\_2**

Value -41,

Could not find a valid environment or file approach in atrshmlog\_attach.

Rational:

You have no valid environment and no flag files set.

## **atrshmlog\_error\_attach\_3**

Value -42,

The cleanup could not successful made atexit in atrshmlog\_attach.

Rational:

There was a problem to make the atexit for the cleanup. Check the number of functions that are used with atexit. If you reach the system limit you have to combine functions to reduce. The module needs at last one atexit slot.

## **atrshmlog\_error\_attach\_4**

Value -43,

The flag file seems to be corrupt in atrshmlog\_attach.

Rational:

The flag file should contain numbers. Check this.

## **atrshmlog\_error\_attach\_5**

Value -44,

No flag file option for this level in atrshmlog\_attach.

Rational:

You gave a flag file in a level where no flag file is allowed.

### **atrshmlog\_error\_attach\_6**

Value -45,

The cleanup could not successful made atexit in atrshmlog\_attach.

Rational:

There was a problem to make the atexit for the cleanup. Check the number of functions that are used with atexit. If you reach the system limit you have to combine functions to reduce. The module needs at last one atexit slot.

### **atrshmlog\_error\_attach\_7**

Value -46,

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

### **atrshmlog\_error\_init\_in\_write\_1**

Value -51,

Buffer allocation failure in atrshmlog\_init\_in\_write.

Rational:

The initialization ended without getting a buffer. This means normally you have an out of memory situation.

Check for threads that do end but not give back the logging buffers to the module. You have two buffers per thread so you can easily get an out of memory if you do start many threads but end them without a stop or turn off.

## **atrshmlog\_error\_write0\_1**

Value -61,

Eventnumber negative in atrshmlog\_write0.

Rational:

Parameter error, check your code.

## **atrshmlog\_error\_write0\_2**

Value -62,

Eventnumber too big in atrshmlog\_write0.

Rational:

Your eventnumber is bigger than the max event locks.

The program needs bigger events and you didn't adjust the module or didn't set the max event locks high enough.

OR: Your environment is wrong and you miss the variable to set the max event locks.

OR: You overwrote the limit flag. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_write0\_3**

Value -63,

Logging off for the thread in atrshmlog\_write0.

Rational:

Your eventnumber is bigger than the max event locks.

The program needs bigger events and you didn't adjust the module or didn't set the max event locks high enough.

OR: Your environment is wrong and you miss the variable to set the max event locks.

OR: You overwrote the limit flag. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_write0\_4**

Value 61,

Buffer full discard in atrshmlog\_write0.

Rational:

You had a buffer full situation and your actual strategy is discard. So the log was aborted.

### **atrshmlog\_error\_write0\_5**

Value 62,

Logging off final in atrshmlog\_write0.

Rational:

The program shuts down so no more logging is allowed.

OR: Someone has overwritten the final flag, shut down the program. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

### **atrshmlog\_error\_write0\_6**

Value 63,

Logging off in area in atrshmlog\_write0.

Rational:

The system wide logging flag has been switched to off.

### **atrshmlog\_error\_write0\_7**

Value -64,

The safeguard is corrupt in atrshmlog\_write0.

Rational:

The buffer you use in the program is corrupt, shut down the program. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

### **atrshmlog\_error\_write0\_8**

Value -65,

The area safeguard is corrupt in atrshmlog\_write0.

Rational:

You have a memory overwrite for the area. Stop logging and shut down that area. At least reinit it.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

### **atrshmlog\_error\_write0\_9**

Value 64,

Logging off in area atrshmlog\_write0.

Rational:

The system wide logging flag has been switched to off.

### **atrshmlog\_error\_write0\_10**

Value -66,

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

### **atrshmlog\_error\_write1\_1**

Value -71,

Size payload is negative in atrshmlog\_write1.

Rational:

Parameter error, check your code.

### **atrshmlog\_error\_write1\_2**

Value -72,

Eventnumber negative in atrshmlog\_write1.

Rational:

Parameter error, check your code.

## **atrshmlog\_error\_write1\_3**

Value -73,

Eventnumber too big in atrshmlog\_write1.

Rational:

Your eventnumber is bigger than the max event locks.

The program needs bigger events and you didn't adjust the module or didn't set the max event locks high enough.

OR: Your environment is wrong and you miss the variable to set the max event locks.

OR: You overwrote the limit flag. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_write1\_4**

Value -74,

Payload too big for logging in atrshmlog\_write1.

Rational:

Your payload is too big for the maximum possible size to log.

## **atrshmlog\_error\_write1\_5**

Value -75,

Logging off for the thread in atrshmlog\_write1.

Rational:

The thread does not log. The init was not successful or someone has switched logging off after.

## **atrshmlog\_error\_write1\_6**

Value 71,

Buffer full discard in atrshmlog\_write1.

Rational:

You had a buffer full situation and your actual strategy is discard. So the log was aborted.

## **atrshmlog\_error\_write1\_7**

Value 72,

Logging off final in atrshmlog\_write1.

Rational:

The program shuts down so no more logging is allowed.

OR: Someone has overwritten the final flag, shut down the program. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_write1\_8**

Value 73,

Logging off in area in atrshmlog\_write1.

Rational:

The system wide logging flag has been switched to off.

## **atrshmlog\_error\_write1\_9**

Value -76,

Payload too big for logging in atrshmlog\_write1.

Rational:

You have hit the maximum size possible.

## **atrshmlog\_error\_write1\_10**

Value -77,

The safeguard is corrupt in atrshmlog\_write1.

Rational:

The buffer you use in the program is corrupt, shut down the program. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_write1\_11**

Value -78,

The area safeguard is corrupt in atrshmlog\_write1.

Rational:

You have a memory overwrite for the area. Stop logging and shut down that area. At least reinit it.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

### **atrshmlog\_error\_write1\_12**

Value 74,

Logging off in area in atrshmlog\_write1.

Rational:

The system wide logging flag has been switched to off.

### **atrshmlog\_error\_write1\_13**

Value -79,

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

### **atrshmlog\_error\_write2\_1**

Value -81,

Size payload is negative in atrshmlog\_write2.

Rational:

Parameter error, check your code.

### **atrshmlog\_error\_write2\_2**

Value -82,

Eventnumber negative in atrshmlog\_write2.

Rational:

Parameter error, check your code.

### **atrshmlog\_error\_write2\_3**

Value -83,

Eventnumber too big in atrshmlog\_write2.

Rational:

Your eventnumber is bigger than the max event locks.

The program needs bigger events and you didn't adjust the module or didn't set the max event locks high enough.

OR: Your environment is wrong and you miss the variable to set the max event locks.

OR: You overwrote the limit flag. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

### **atrshmlog\_error\_write2\_4**

Value -84,

Payload too big for logging in atrshmlog\_write2.

Rational:

Your payload is too big for the maximum possible size to log.

### **atrshmlog\_error\_write2\_5**

Value -85,

Logging off for the thread in atrshmlog\_write2.

Rational:

The thread does not log. The init was not successful or someone has switched logging off after.

### **atrshmlog\_error\_write2\_6**

Value 81,

Buffer full discard in atrshmlog\_write2.

Rational:

You had a buffer full situation and your actual strategy is discard. So the log was aborted.

### **atrshmlog\_error\_write2\_7**

Value 82,

Logging off final in atrshmlog\_write2.

Rational:

The program shuts down so no more logging is allowed.

OR: Someone has overwritten the final flag, shut down the program. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

### **atrshmlog\_error\_write2\_8**

Value 83,

Logging off in area in atrshmlog\_write2.

Rational:

The system wide logging flag has been switched to off.

### **atrshmlog\_error\_write2\_9**

Value -86,

Payload too big for logging in atrshmlog\_write2.

Rational:

You have hit the maximum size possible.

### **atrshmlog\_error\_write2\_10**

Value -87,

The safeguard is corrupt in atrshmlog\_write2.

Rational:

The buffer you use in the program is corrupt, shut down the program. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_write2\_11**

Value -88,

The area safeguard is corrupt in atrshmlog\_write2.

Rational:

You have a memory overwrite for the area. Stop logging and shut down that area. At least reinit it.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

## **atrshmlog\_error\_write2\_12**

Value 84,

Logging is off in area in atrshmlog\_write2.

Rational:

The system wide logging flag has been switched to off.

## **atrshmlog\_error\_write2\_13**

Value -89,

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

## **atrshmlog\_error\_area\_version\_1**

Value -91,

The area is not connected in atrshmlog\_get\_area\_version.

Rational:

No valid connect was made – did you forget to attach ?

OR: Possible is an overwrite inside the process that nulled the pointer variable.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_area\_count\_1**

Value -101,

The area is not connected in atrshmlog\_get\_area\_count.

Rational:

No valid connect was made – did you forget to attach ?

OR: Possible is an overwrite inside the process that nulled the pointer variable.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_area\_ich\_habe\_fertig\_1**

Value -111,

The area is not connected in atrshmlog\_set\_area\_ich\_habe\_fertig.

Rational:

No valid connect was made – did you forget to attach ?

OR: Possible is an overwrite inside the process that nulled the pointer variable.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_get\_event\_1**

Value -121,

The index is out of range in atrshmlog\_get\_event.

Rational:

You have given an event false.

OR: The program needs bigger events and you didn't adjust the module or didn't set the max event locks high enough.

OR: Your environment is wrong and you miss the variable to set the max event locks.

OR: You overwrote the limit flag. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_get\_logging\_1**

Value 131,

The area is not connected in atrshmlog\_get\_logging.

Rational:

No valid connect was made – did you forget to attach ?

OR: Possible is an overwrite inside the process that nulled the pointer variable.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_get\_logging\_2**

Value 132,

Area logging flag is off in atrshmlog\_get\_logging.

Rational:

That is OK. Someone hit the atrshmlogoff or did switch off. Some programs do that too.

OR: You have a memory overwrite in the area. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

## **atrshmlog\_error\_get\_logging\_3**

Value 133,

Logging final is off in atrshmlog\_get\_logging.

Rational:

You have switched that flag on. So no more logging is possible.

## **atrshmlog\_error\_get\_logging\_4**

Value 134,

Process logging is off in atrshmlog\_get\_logging.

Rational:

No error. Your process has switched the flag to off.

## **atrshmlog\_error\_create\_1**

Value -141,

The ipc key was out of range in atrshmlog\_create.

Rational:

This can normally only be a parameter error – you cannot use negative numbers. Or you have made an error in the parameter handling in the program.

OR: For the mingw you tried less than 1 or more than 32.

## **atrshmlog\_error\_create\_2**

Value -142,

The buffer count is too low in atrshmlog\_create.

Rational:

You tried to use a count lower then the limit. Is your parameter handling correct ?

## **atrshmlog\_error\_create\_3**

Value -143,

The buffer count is too high in atrshmlog\_create.

Rational:

You tried to use a count higher then the limit. Is your parameter handling correct ?

## **atrshmlog\_error\_create\_4**

Value -144,

The connect failed in low level call in atrshmlog\_create.

Rational:

The OS operation failed. Try a smaller area. Check for access rights. Check the errno value if your OS supports it.

## **atrshmlog\_error\_init\_shm\_1**

Value -151,

The area is NULL in atrshmlog\_init\_shm\_log.

Rational:

No valid connect was made – did you forget to attach ?

OR: Possible is an overwrite inside the process that nulled the pointer variable.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

### **atrshmlog\_error\_init\_shm\_2**

Value -152,

The shared memory id is different for process and area in atrshmlog\_init\_shm\_log.

Rational:

You have the wrong environment to use that area.

OR: It is not initialized at all.

### **atrshmlog\_error\_init\_shm\_3**

Value -153,

The low level init failed in atrshmlog\_init\_shm\_log.

Rational:

For a buffer an init failed. No further info.

OR: You have eventually a problem with the shared memory – can be a problem of wrong size calculations. Check the buffer counts used.

### **atrshmlog\_error\_read\_1**

Value -161,

The area is NULL in atrshmlog\_read.

Rational:

No valid connect was made – did you forget to attach ?

OR: Possible is an overwrite inside the process that nulled the pointer variable.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

## **atrshmlog\_error\_read\_2**

Value -162,

Buffer index negative in atrshmlog\_read.

Rational:

Parameter error in your program.

## **atrshmlog\_error\_read\_3**

Value -163,

The buffer index is too big in atrshmlog\_read.

Rational:

The area has not that many buffers – did you get the area buffer count for the index limit or did you get it from somewhere else ?

OR: You connected to an area that was created earlier with another count of buffers.

OR: The count in the area has been overwritten. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

## **atrshmlog\_error\_read\_4**

Value -164,

The area safeguard was corrupt in atrshmlog\_read.

Rational:

You have a memory overwrite for the area. Stop logging and shut down that area. At least reinit it.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

## **atrshmlog\_error\_read\_5**

Value -165,

The buffer size in shared memory was too big in atrshmlog\_read.

Rational:

That means you have the wrong maximum size. Stop logging. You have different version tools for logging and for the reader, so you work with different module versions – at least one was changed but not for the version.

OR: It can also mean you have a memory overwrite for the size in the area. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

### **atrshmlog\_error\_read\_6**

Value -166,

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

### **atrshmlog\_error\_read\_fetch\_1**

Value -171,

The area is NULL in atrshmlog\_read\_fetch.

Rational:

No valid connect was made – did you forget to attach ?

OR: Possible is an overwrite inside the process that nulled the pointer variable.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

### **atrshmlog\_error\_read\_fetch\_2**

Value -172,

The area safeguard was corrupt in atrshmlog\_read\_fetch.

Rational:

You have a memory overwrite for the area. Stop logging and shut down that area. At least reinit it.

Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

## **atrshmlog\_error\_read\_fetch\_3**

Value 171,

No buffer to be processed in atrshmlog\_read\_fetch.

Rational:

This is OK. Most the time you will encounter in a low traffic scenario this fact. Simply skip the rest you plan for a buffer to do here and start next test.

## **atrshmlog\_error\_read\_fetch\_4**

Value -173,

The buffer size in shared memory was too big in atrshmlog\_read\_fetch.

Rational:

That means you have the wrong maximum size. Stop logging. You have different version tools for logging and for the reader, so you work with different module versions – at least one was changed but not for the version.

OR: It can also mean you have a memory overwrite for the size in the area. Set up a test to stabilize the overwrite then. Take a debugger check macro if possible and hit down the overwrite function.

OR: It can mean you have a serious synchronization problem and need the fences.

## **atrshmlog\_error\_read\_fetch\_5**

Value 172,

The fetch gave a buffer has size 0 in shared memory in atrshmlog\_read\_fetch.

Rational:

The operation simply didn't find a valid buffer to transfer. Could be an empty list or a size 0 buffer itself.

No major issue.

## **atrshmlog\_error\_read\_fetch\_6**

Value -174,

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

### **atrshmlog\_error\_verify\_1**

Value -181,

The check found that the Area was not attached in atrshmlog\_verify.

Rational:

This can be a wrong ID in the environment.

Or: The access is not given for the process and the owner.

### **atrshmlog\_error\_verify\_2**

Value -182,

The check found the shared memory id differs to process shared memory id in atrshmlog\_verify.

Rational:

You have the wrong environment to use that area.

OR: It is not initialized at all.

### **atrshmlog\_error\_verify\_3**

Value -183,

The check found an area safeguard was corrupted in atrshmlog\_verify.

Rational:

Best is to stop logging. Shutdown that area, make a new or at least reinit it.

Set up a test to stabilize the overwrite then.

Take a debugger check macro if possible and hit down the overwrite function.

### **atrshmlog\_error\_verify\_4**

Value -184,

The check found a version mismatch for area and module code in atrshmlog\_verify.

Rational:

That could mean you use different versions of the module.

OR: It could mean you have a serious memory synchronization problem between processes on the system. Switch the fences on and retry.

### **atrshmlog\_error\_verify\_5**

Value -185,

The check found a area buffer state was corrupt in atrshmlog\_verify.

Rational:

Best is to stop logging. Shutdown that area, make a new or at least reinit it.

Set up a test to stabilize the overwrite then.

Take a debugger check macro if possible and hit down the overwrite function.

### **atrshmlog\_error\_verify\_6**

Value -186

The check found for a buffer that the safeguard was corrupted in atrshmlog\_verify.

Rational:

Best is to stop logging. Shutdown that area, make a new or at least reinit it.

Set up a test to stabilize the overwrite then.

Take a debugger check macro if possible and hit down the overwrite function.

### **atrshmlog\_error\_buffer\_slave\_1**

Value -190,

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

### **atrshmlog\_error\_get\_strategy\_1**

Value -191,

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

### **atrshmlog\_error\_set\_strategy\_1**

Value -192,

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

### **atrshmlog\_error\_get\_autoflush\_1**

Value -200

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

### **atrshmlog\_error\_set\_autoflush\_1**

Value -201

Pthread specific buffer not available.

Rational:

We could not get the pthread specific key up. That means no logging. This can only happen for the platforms without thread local storage.

## Statistics

The module has build in a counter array. Its made of atomics. So the counters are correct if you don't overwrite the thing. For the write the statistics is in thread local memory to circumvent interaction of threads when logging.

If you use them you can get an int array with them with the get statistics function.

There is a helper to comment them. It depends on a special format for the output. See atrshmlogstat.

We give here the enum for the counter positions and a small rational for its meaning.

The enum is atrshmlog\_counter.

### **atrshmlog\_counter\_time\_low**

Value 0,

We deliver in the get function the actual click time low part.

Rational:

This is the 32 bit low part of the click time when we take the statistics.

### **atrshmlog\_counter\_time\_high**

Value 1,

We deliver in the get function the actual click time high part

Rational:

This is the 32 bit high part of the click time when we take the statistics.

### **atrshmlog\_counter\_attach**

Value 2,

The number of calls to atrshmlog\_attach()

Rational:

You should check this is only a 1.

### **atrshmlog\_counter\_get\_raw**

Value 3,

The number of calls to `atrshmlog_il_get_raw_buffers()`

Rational:

The number of allocs of dynamic memory. This should be small, best 0. Check if you have a high count the prealloc and the count so static buffers for the module and the number of threads. Check also if you do not give back buffers via turn off or stop.

### **atrshmlog\_counter\_free**

Value 4,

The number of calls to `atrshmlog_free()`

Rational:

You should see here a high number if you start many threads and they are turned off or stop.

### **atrshmlog\_counter\_alloc**

Value 5,

The number of calls to `atrshmlog_alloc()`

Rational:

The number of buffers you get from the available list. You can see if you start many non logging threads that this is much smaller than the number of threads times the number of buffers per thread.

### **atrshmlog\_counter\_dispatch**

Value 6,

The number of calls to `atrshmlog_dispatch_buffer()`

Rational:

The number of buffers you put on the full list. This should be the number you see in mem to shm if you don't force empty buffers to be moved.

### **atrshmlog\_counter\_mem\_to\_shm**

Value 7,

The number of calls to `atrshmlog_transfer_mem_to_shm()` Rational:

Rational:

The number of buffers you try to move.

### **atrshmlog\_counter\_mem\_to\_shm\_doit**

Value 8,

When `atrshmlog_transfer_mem_to_shm()` actually starts to transfer

Rational:

If you really do a transfer of a buffer. This should be for all clients the same number of buffers in the file system or you have a shared memory problem.

### **atrshmlog\_counter\_mem\_to\_shm\_full**

Value 9,

When `atrshmlog_transfer_mem_to_shm()` runs into a full shm buffer system and has to wait

Rational:

You have a slow reader or the system is busy.

### **atrshmlog\_counter\_create\_slave**

Value 10,

The number of calls to `atrshmlog_create_slave()`

Rational:

This can show you if you start slaves abnormal in another place.

### **atrshmlog\_counter\_stop**

Value 11,

The number of calls to `atrshmlog_stop()`

Rational:

The number you stop logging for a thread. This and turn off should together be comparable to the number of threads you end or you loose log buffers.

## **atrshmlog\_counter\_write0**

Value 12,

The number of calls to atrshmlog\_write0()

Rational:

The number of write0 in your program.

## **atrshmlog\_counter\_write0\_abort1**

Value 13,

When atrshmlog\_write0() exits because of error

Rational:

Parameter error negative eventnumber.

## **atrshmlog\_counter\_write0\_abort2**

Value 14,

When atrshmlog\_write0() exits because of error

Rational:

Parameter error eventnumber too big.

## **atrshmlog\_counter\_write0\_abort3**

Value 15,

When atrshmlog\_write0() exits because of error

Rational:

Logging was shut off for this thread or a init error.

## **atrshmlog\_counter\_write0\_abort4**

Value 16,

When atrshmlog\_write0() exits because of error

Rational:

Error in init.

## **atrshmlog\_counter\_write0\_discard**

Value 17,

When `atrshmlog_write0()` exits because strategy discard

Rational:

Number of logs you lost in a buffer full situation because your strategy is discard.

## **atrshmlog\_counter\_write0\_wait**

Value 18,

When `atrshmlog_write0()` waits because strategy

Rational:

Number of buffer full waits you encounter. Should be as small as possible.

## **atrshmlog\_counter\_write0\_adaptive**

Value 19,

When `atrshmlog_write0()` waits because strategy

Rational:

Number of buffer full and adaptive waits. Should be as small as possible.

## **atrshmlog\_counter\_write0\_adaptive\_fast**

Value 20,

When `atrshmlog_write0()` waits because strategy

Rational:

Number of buffer full and adaptive waits. Should be as small as possible.

## **atrshmlog\_counter\_write0\_adaptive\_very\_fast**

Value 21,

When `atrshmlog_write0()` waits because strategy

Rational:

Number of buffer full and adaptive waits. Should be as small as possible.

### **atrshmlog\_counter\_write\_safeguard**

Value 22,

When `atrshmlog_write0()` exits because safeguard error

Rational:

The number of corrupt buffers. Should be 0. Check program integrity else.

### **atrshmlog\_counter\_write\_safeguard\_shm**

Value 23,

When `atrshmlog_write0()` exits because safeguard error

Rational:

Number of corrupt area. Should be 0 . Check synchronization problems and fences .

### **atrshmlog\_counter\_write1**

Value 24,

The number of calls to `atrshmlog_write1()`

Rational:

The number of write1 for your program.

### **atrshmlog\_counter\_write1\_abort1**

Value 25,

When `atrshmlog_write1()` exits because of error

Rational:

Parameter error negative eventnumber.

### **atrshmlog\_counter\_write1\_abort2**

Value 26,

When `atrshmlog_write1()` exits because of error

Rational:

Parameter error eventnumber too big. Check code and max locks.

### **atrshmlog\_counter\_write1\_abort3**

Value 27,

When `atrshmlog_write1()` exits because of error

Rational:

Logging was shut off for this thread or a init error.

### **atrshmlog\_counter\_write1\_abort4**

Value 28,

When `atrshmlog_write1()` exits because of error

Rational:

Error in init.

### **atrshmlog\_counter\_write1\_discard**

Value 29,

When `atrshmlog_write1()` exits because strategy discard

Rational:

Number of logs you lost in a buffer full situation because your strategy is discard.

### **atrshmlog\_counter\_write1\_wait**

Value 30,

When `atrshmlog_write1()` waits because strategy

Rational:

Number of buffer full waits you encounter. Should be as small as possible.

## **atrshmlog\_counter\_write1\_adaptive**

Value 31,

When `atrshmlog_write1()` waits because strategy

Rational:

Number of buffer full and adaptive waits. Should be as small as possible.

## **atrshmlog\_counter\_write1\_adaptive\_fast**

Value 32,

When `atrshmlog_write1()` waits because strategy

Rational:

Number of buffer full and adaptive waits. Should be as small as possible.

## **atrshmlog\_counter\_write1\_adaptive\_very\_fast**

Value 33,

When `atrshmlog_write1()` waits because strategy

Rational:

Number of buffer full and adaptive waits. Should be as small as possible.

## **atrshmlog\_counter\_write1\_abort5**

Value 34,

When `atrshmlog_write1()` exits because of error

Rational:

Parameter error, size of payload negative. Check program code.

## **atrshmlog\_counter\_write1\_abort6**

Value 35,

When `atrshmlog_write1()` exits because of error

Rational:

Payload too big, can never fit. Check program code.

### **atrshmlog\_counter\_write1\_abort7**

Value 36,

When atrshmlog\_write1() exits because of error

Rational:

Payload too big for that buffer. Check your reduce of size and your need for log size.

### **atrshmlog\_counter\_write2**

Value 37,

The number of calls to atrshmlog\_write2()

Rational:

The number of write2 for your program.

### **atrshmlog\_counter\_write2\_abort1**

Value 38,

When atrshmlog\_write2() exits because of error

Rational:

Parameter error negative eventnumber.

### **atrshmlog\_counter\_write2\_abort2**

Value 39,

When atrshmlog\_write2() exits because of error

Rational:

Parameter error eventnumber too big. Check code and max locks.

### **atrshmlog\_counter\_write2\_abort3**

Value 40,

When atrshmlog\_write2() exits because of error

Rational:

Logging was shut off for this thread or a init error.

### **atrshmlog\_counter\_write2\_abort4**

Value 41,

When atrshmlog\_write2() exits because of error

Rational:

Error in init.

### **atrshmlog\_counter\_write2\_discard**

Value 42,

When atrshmlog\_write2() exits because strategy discard

Rational:

Number of logs you lost in a buffer full situation because your strategy is discard.

### **atrshmlog\_counter\_write2\_wait**

Value 43,

When atrshmlog\_write2() waits because strategy

Rational:

Number of buffer full waits you encounter. Should be as small as possible.

### **atrshmlog\_counter\_write2\_adaptive**

Value 44,

When atrshmlog\_write2() waits because strategy

Rational:

Number of buffer full and adaptive waits. Should be as small as possible.

### **atrshmlog\_counter\_write2\_adaptive\_fast**

Value 45,

When `atrshmlog_write2()` waits because strategy

Rational:

Number of buffer full and adaptive waits. Should be as small as possible.

### **atrshmlog\_counter\_write2\_adaptive\_very\_fast**

Value 46,

When `atrshmlog_write2()` waits because strategy

Rational:

Number of buffer full and adaptive waits. Should be as small as possible.

### **atrshmlog\_counter\_write2\_abort5**

Value 47,

When `atrshmlog_write2()` exits because of error

Rational:

Parameter error, size of payload negative. Check program code.

### **atrshmlog\_counter\_write2\_abort6**

Value 48,

When `atrshmlog_write2()` exits because of error

Rational:

Payload too big, can never fit. Check program code.

## **atrshmlog\_counter\_write2\_abort7**

Value 49,

When `atrshmlog_write2()` exits because of error

Rational:

Payload too big for that buffer. Check your reduce of size and your need for log size.

## **atrshmlog\_counter\_set\_slave\_count**

Value 50,

The number of calls to `atrshmlog_set_f_list_buffer_slave_count()`

Rational:

IF you need more slaves you should see this is used.

## **atrshmlog\_counter\_set\_clock\_id**

Value 51,

The number of calls to `atrshmlog_set_clock_id()`

Rational:

You should see if the clock is changed. Can be a problem with the op code on the CPU.

## **atrshmlog\_counter\_slave\_off**

Value 52,

The number of calls to `atrshmlog_set_f_list_buffer_slave_run_off()`

Rational:

If you get in buffer full situations you should check if the slaves were shut off.

## **atrshmlog\_counter\_set\_event\_locks**

Value 53,

The number of calls to `atrshmlog_set_event_locks_max()`

Rational:

IF you need bigger event numbers and get errors for events you should check this is used.

### **atrshmlog\_counter\_set\_buffer\_size**

Value 54,

The number of calls to `atrshmlog_set_buffer_infosize()`

Rational:

If you have too small buffers you should see if you have reduced them.

### **atrshmlog\_counter\_set\_wait\_slaves\_on**

Value 55,

The number of calls to `atrshmlog_set_wait_for_slaves_on()`

Rational:

If you hang in cleanup and you see this is not 0 you should check your slaves are all alive and if you kill them you have maintained the slave count.

### **atrshmlog\_counter\_set\_wait\_slaves\_off**

Value 56,

The number of calls to `atrshmlog_set_wait_for_slaves_off()`

Rational:

You should know if the wait as set off.

### **atrshmlog\_counter\_set\_slave\_wait**

Value 57,

The number of calls to `atrshmlog_set_f_list_buffer_slave_wait()`

Rational:

IF you have a high CPU load for slaves you should see if you change the wait.

## **atrshmlog\_counter\_set\_prealloc\_count**

Value 58,

The number of calls to `atrshmlog_set_prealloc_buffer_count()`

Rational:

IF you need another prealloc count you should know this.

## **atrshmlog\_counter\_set\_thread\_fence**

Value 59,

The number of calls to `atrshmlog_set_thread_fence()`

Rational:

IF you need fences you should see here a small number.

## **atrshmlog\_counter\_create**

Value 60,

The number of calls to `atrshmlog_create()`

Rational:

If you create buffers you should know this.

## **atrshmlog\_counter\_create\_abort1**

Value 61,

The `atrshmlog_create()` exits because of error

Rational:

Parameter error ipc key, check your code or environment.

## **atrshmlog\_counter\_create\_abort2**

Value 62,

The `atrshmlog_create()` exits because of error

Rational:

Parameter error for count, check your code or environment.

### **atrshmlog\_counter\_create\_abort3**

Value 63,

The `atrshmlog_create()` exits because of error

Rational:

Parameter error for your count, check your code or environment.

### **atrshmlog\_counter\_create\_abort4**

Value 64,

The `atrshmlog_create()` exits because of error

Rational:

OS error. Check the maximum available size and access rights.

### **atrshmlog\_counter\_delete**

Value 65,

The number of calls to `atrshmlog_delete()`

Rational:

If you destroy the buffer you should know this.

### **atrshmlog\_counter\_cleanup\_locks**

Value 66,

The number of calls to `atrshmlog_cleanup_locks()`

Rational:

If you destroy the area you should know this.

### **atrshmlog\_counter\_init\_shm**

Value 67,

The number of calls to `atrshmlog_init_shm_log()`

Rational:

If you do the init internal you should be sure not to do it more than create.

## **atrshmlog\_counter\_read**

Value 68,

The number of calls to `atrshmlog_read()`

Rational:

Old function, should not be used in normal cases.

## **atrshmlog\_counter\_read\_doit**

Value 69,

The `atrshmlog_read()` transfers a buffer

Rational:

Old function, should not be used in normal cases.

## **atrshmlog\_counter\_read\_fetch**

Value 70,

The number of calls to `atrshmlog_read_fetch()`

Rational:

The total number of read try's. If this is much more than the doit count you should give the reader a higher wait time. Its wasting CPU else.

## **atrshmlog\_counter\_read\_fetch\_doit**

Value 71,

The `atrshmlog_read_fetch()` transfers a buffer

Rational:

Number of transferred buffers with a size greater 0 – should be identical to the count of bin files in the programs write file tree.

## **atrshmlog\_counter\_verify**

Value 72,

The number of calls to `atrshmlog_verify()`

Rational:

IF you have a high verify count you should check if someone has made a sick joke and does always a verify before a log. That's a performance no go.

### **atrshmlog\_counter\_logging\_process\_on**

Value 73,

The number of calls to `atrshmlog_set_logging_process_on()`

Rational:

IF you have less than off you should check why you shut off so much.

### **atrshmlog\_counter\_logging\_process\_off**

Value 74,

The number of calls to `atrshmlog_set_logging_process_off()`

Rational:

If you have more than one you should check this if you loose valuable log.

### **atrshmlog\_counter\_set\_strategy**

Value 75,

The number of calls to `atrshmlog_set_strategy()`

Rational:

If you have a count here comparable to the number of threads you should consider to change the default instead for the program.

### **atrshmlog\_counter\_set\_strategy\_process**

Value 76,

The number of calls to `atrshmlog_set_strategy_process()`

Rational:

IF you have here more than one you should consider to rethink what is your program wide default.

## **atrshmlog\_counter\_set\_event**

Value 77,

The number of calls to `atrshmlog_set_event_on()`

Rational:

IF you have a high count here check why you cannot live with the default init you have choose for the events.

## **atrshmlog\_counter\_set\_env\_prefix**

Value 78,

The number of calls to `atrshmlog_set_event_off()`

Rational:

If you encounter problems with use of environment or flag files it could be you change the prefix.

## **atrshmlog\_counter\_exit\_cleanup**

Value 79,

The number of calls to `atrshmlog_exit_cleanup()`

Rational:

IF there are more than one its likely you have a problem with atexit functions.

## **atrshmlog\_counter\_flush**

Value 80,

The number of calls to `atrshmlog_flush()`

Rational:

If you often flush you slow down the logging and so the threads.

## **atrshmlog\_counter\_logging\_process\_off\_final**

Value 81,

The number of calls to `atrshmlog_set_logging_process_off_final()`

Rational:

IF you have logs not receiving its possible you have stopped logging else in your system.

### **atrshmlog\_counter\_turn\_logging\_off**

Value 82,

The number of calls to `atrshmlog_turn_logging_off()`

Rational:

You switch threads off in your program. That's OK if the thread does not do itself a stop. So take the number of threads you create, and check the number of turn off you make. Should be comparable.

If not you could have memory problems if the threads log.

### **atrshmlog\_counter\_init\_in\_advance\_on**

Value 83,

The number of calls to `atrshmlog_init_buffers_in_advance_on()`

Rational:

If you switch the advance on you loose some time with memset and calloc instead of a malloc.

### **atrshmlog\_counter\_init\_in\_advance\_off**

Value 84,

The number of calls to `atrshmlog_init_buffers_in_advance_off()`

Rational:

If you switch the advance off you can encounter problems on mingw and also cygwin. So check why you switch it off.

### **atrshmlog\_counter\_reuse\_thread\_buffers**

Value 85,

The number of calls to `atrshmlog_ruswe_thread_buffers()`

Rational:

If you reuse buffers you need to know how many are.

## **atrshmlog\_counter\_set\_autoflush**

Value 86,

The numbers you set the autoflush for process flag.

Rational:

If you switch the autoflush you need to know this.

## **atrshmlog\_counter\_fence\_alarm\_1**

Value 87,

The numbers a checksum error is found in the slave.

Rational:

If you encounter consistency problems you need to know where they happen. So this is one spot you can defect them.

## **atrshmlog\_counter\_fence\_alarm\_2**

Value 88,

The numbers a checksum error is found in the read\_fetch.

Rational:

If you encounter consistency problems you need to know where they happen. So this is one spot you can defect them.

## **Thread local statistics**

To get the thread local statistic it was transferred with the log buffer via reader into conventional space. You have a sequence counter so the latest version with the highest sequence is the real thing. Check the output of the converter for it.

The converter can deliver the statistics if you use three parameters. The third is then a file that the converter creates to get the statistics in text form.

You then can use the atrshmlogstat if you want to see for the meaning.

Its only the vital runtime dependent stuff. The error counts are still in the main statistics.

## **Strategy**

There is no unlimited number of buffers for a thread. So we have to accept that it is possible that we have all buffers to log full when we enter the check for a free one.

So we have to wait.

OK, that is no perfect world, and so waiting is also a bad thing, but we simply have to.

When it comes down to this we have some options how to do this.

So we can set the default for the process, and then every thread gets that in its init and uses from then on the own value.

If you have a different behavior you need you can set the thread strategy. If you think the program is better served with it you can set a process strategy that differs too.

So here are the strategy values. Its an enum after all, but you can use the int instead.

The enum is atrshmlog\_strategy

### **atrshmlog\_strategy\_discard**

Value 0,

Rational :

If you have to do things very fast, but also need a log if possible, but cannot wait the rough 250000 up clicks in case you have a buffer full you can discard the log. This will cost only the time to find that you have to – rough 20 to 30 clicks – and that its.

So if you have for example a garbage collector and you need it to run but want have info if possible the discard would be a good strategy.

### **atrshmlog\_strategy\_spin\_loop**

Value 1,

Rational:

If you have a single or near single thread application, and you want to log at all cost, but need best response time – then the spin lock is an option. It will burn a CPU then. But you get a log – no discard – and you get the best response possible.

### **atrshmlog\_strategy\_wait**

Value 2,

Rational:

This is a simple default. You estimate a wait time. And so you set this in here. There is a default and you can set it if you need different.

### **atrshmlog\_strategy\_adaptive**

Value 3,

Rational:

This is using the last transfer time for a buffer. So you have the last for it in the module. You use a rough calculation to get it to nanoseconds. Then you scale it down with the number of buffers. So you get an average wait time this way – assuming its a more or less sequential thing.

This is a self adjusting thing, so it can be very good or get bad in case you have a peek time far from usual. But it is calculated always against the very last time you encounter. So its most up to date, and if you encounter heavy CPU loads its perhaps the best of all.

### **atrshmlog\_strategy\_adaptive\_fast**

Value 4,

Rational:

See the adaptive first. Now for this on the difference. You encounter peaks and want to reduce the effect. So this one make an additional cut down of factor 2. It comes faster back to the test this way.

### **atrshmlog\_strategy\_adaptive\_very\_fast**

Value 5,

Rational:

See the adaptive first. Now for this on the difference. You encounter heavy peaks and want to reduce the effect. So this one make an additional cut down of factor 10. It comes much faster back to the test this way.

## Environment setting

The module makes use of at least one environment variable at the initialization.

The name is determined by two things. The first is the prefix buffer content. The second is the define ATRSHMLOGENV\_SUFFIX.

The prefix is first an empty buffer. No content. Its max 128 chars. The content can be set in advance before first use with a setter. So there you are then with your own prefix. Next is to first try to read in a value. If not already set then the prefix is first default set with the define ATRSHMLOG\_ENV\_PREFIX.

Then a first lookup is made to find if there is an environment variable with that name. If there is one, its value is used to refill the prefix buffer.

This sounds a bit overdue.

You have a default and that can be set in the program before first use – so you are in for your name here.

OK. You can also change the define – its open source after all.

If you cannot do the setter – perhaps its not you that makes the attach, instead another company programmer in his code – or even a contractor – then you can also likely not change the define for the “No changes to alien code” policy ....

Then you can resort to the setting of the ATRSHMLOG variable and you are in with your naming.

So from this moment on all look ups are made with the prefix and the suffix is simply appended.

The prefix cannot be changed after this.

For the first lookup in the module its natural to be the lookup for the shared memory buffer's ID of the system. If its the mingw you have here the index in the hidden naming array, for the posix systems the shared memory id of the created buffer.

After that access is made you get a check for it. If its not there the thing tries to switch to the so called flag files. And that is then used for all accesses.

They have the same name as the environment variable – so all capital letters. And a suffix of “.TXT”. So you can set the values in case you cannot use the environment variables by using the flag files. The files are read in for most cases and the content is expected to be the first and often only thing.

A plain number.

So what follows here is the list of environment variables that the module uses. The list is also spanning the reader. So you should be able to set the configuration of the module with it without trying to check the source code.

There are most the time simple settings, a flag thing with 0 and 1 or a simple number.

The exception is the event setting stuff. This uses a list approach. We use here the default names for the full variable. The prefix is ATRSHMLOG as you can see in the internal header.

## **ATRSHMLOG**

Value : The new prefix.

Define : ATRSHMLOG\_ENV\_PREFIX

Rational:

You can redefine the prefix this way without changing the code. Set ATRSHMLOG=hugo for getting hugo to be the new prefix.

This works only for the first access to the prefix with atrshmlog\_get\_env.

After this access the resulting prefix is fix and cannot be changed.

## **ATRSHMLOG\_ID**

Value : A number that defines the used shared memory buffer for attach.

Define: ATRSHMLOGENV\_SUFFIX

Rational:

We use an environment here to get in the attach the ID of the shared memory buffer. In case of mingw the index for the name array.

The ID is on all systems I know a positive number. It can be small like 65536 in cygwin or something bigger in the rest of the posix systems.

## **ATRSHMLOG\_COUNT**

Value: A number.

Define : ATRSHMLOG\_BUFFER\_COUNT\_SUFFIX

Rational:

The number of buffers you have in the area. We need this value only for mingw. Its an interim solution for now.

Some of the programs that use a parameter for the count of buffers have been altered to use it too. But not all. So check the program code in doubt.

Area buffers are rough half an MB in size. So on many systems you are limited for the total size of the area. Keep this in mind if you try bigger numbers.

## **ATRSHMLOG\_INIT\_IN\_ADVANCE**

Value: A flag 0 or 1.

Define : ATRSHMLOGINITINADVANCESUFFIX

Rational:

You can prepare the buffers in advance with a memset 0 and the dynamic allocated buffers with a calloc. This helps to shift the access time from logging to the initialization phase of the buffers. For the static buffers its done in attach. For the dynamic allocated buffers in the raw get.

## **ATRSHMLOG\_STRATEGY**

Value: A number from 0 to max for the enum's of atrshmlog\_strategy.

Define: ATRSHMLOGSTRATEGYSUFFIX

Rational:

You can set the default strategy for the process in attach this way.

The strategy is then used in the thread local init to init the thread strategy.

## **ATRSHMLOG\_STRATEGY\_WAIT\_TIME**

Value : A number ( 100000 at best, more 250000 or even 750000).

Define : ATRSHMLOGSTRATEGYWAITTIMESUFFIX

Rational:

For the wait time in nanos you use this value as default. IF you use the wait strategy for the buffer full situation you can set it else.

If you use the adaptive the default click time is set to this, but normally the first real transfer wins over it – its only there to circumvent a possible 0 value...

## **ATRSHMLOG\_DELIMITER\_VALUE**

Value : A number resulting in a new C char value from 0 to 255 at least.

Define: ATRSHMLOGDELIMITERSUFFIX

Rational:

You can set the delimiter char for the atrshmlog\_write2 function for the concatenation of the argv

strings.

## **ATRSHMLOG\_EVENT\_COUNT\_MAX**

Value : A number ( at least 10000 )

Define : ATRSHMLOGEVENTCOUNTSUFFIX

Rational:

You can set a new max size for the event flag buffer. This is also the maximum of the valid event values.

If you need a bigger event flag buffer and cannot change the default the this a thing you can use – do it as an interim. Your program will likely have problems when you forget to set it if you made bigger event numbers.

## **ATRSHMLOG\_BUFFER\_SIZE**

Value: A number from 16 K up to the max value.

Define: ATRSHMLOGBUFFER\_INFOSIZE\_SUFFIX

Rational:

The buffer size for the payload log is a define and fix for the static buffers. For the dynamic buffers it can be reduced.

If you have a program need for dynamic allocated buffers and you don't need them to have big log entry's you can reduce this to spare memory. Your dynamic allocated buffers will be of this size for the payload.

So this is more an interim. Only if you cannot switch to static buffers its needed. And only to reduce.

## **ATRSHMLOG\_PREALLOC\_COUNT**

Value: A number.

Define: ATRSHMLOG\_INIT\_PREALLOC\_COUNT\_SUFFIX

Rational:

The number of buffers to fetch in the low level allocation for the dynamic allocation call. So you only pay with one alloc for this many buffers.

But be sure you first tried the static buffer solution – that's always preferable.

And keep in mind that big allocs take time – you will have a wait for the first thread that has to do the thing.

## **ATRSHMLOG\_SLAVE\_WAIT\_NANOS**

Value: A number.

Define: ATRSHMLOG\_INIT\_BUFFER\_SLAVE\_WAIT\_SUFFIX

Rational:

The number of nanos we wait if the slave finds no buffer on the full list. So for this time the process can put buffers on the list, but the slave is simply off.

This reduces CPU load in case you have a low throughput logging.

Times in order of a million could be too big, so try something in range of the memcpy times for the buffers. That should be OK.

## **ATRSHMLOG\_SLAVE\_COUNT**

Value: A number.

Define : ATRSHMLOG\_SLAVE\_COUNT\_SUFFIX

Rational:

We start in the attach and if we encounter a fork clone in the first write the slave threads. So we can set this to a reasonable number.

For a program that does only some logging and then ends normal in exit you can set it to 0 so no slave is active and the cleanup alone transfers the log – spares most CPU time.

The default is to set it to 1 for a low throughput program. It cost rough 1.5 million clicks to fill a log buffer with small payload logs – say 10 to 15 byte per log. So its about 8 to 10 threads when a slave can become to slow to do the transfer to memory for a fast system. For a higher number of threads or a slow system can be different.

So you can set the number that is used there – and only there. Setting it after has no effect. You have then to start them by yourself with the create slave function.

## **ATRSHMLOG\_WAIT\_FOR\_SLAVES\_ON**

Value: A flag 0 or 1.

Define : ATRSHMLOG\_WAIT\_FOR\_SLAVES\_SUFFIX

Rational:

In the cleanup at exit of a normal ending program you can wait for the slaves to stop.

To do this the cleanup switches the slave run flag off. So the slaves – if active – will hit this in the next iteration. Most likely this is in some microseconds after you made the flag.

In case you want to wait for the cleanup then for the slaves to finish you can set this flag.

Then the cleanup waits till the number of slaves is 0 – but this has a caveat: the number is only maintained for regular ending slaves.

So if you play games with slaves, like kill the thread or similar , you have to maintain the number with a helper function.

If you set the flag but have not ending the threads or a false count the thing will hang here. And so you simply loose the log that is still in the buffers when you have to kill the program.

So consider be warned.

## **ATRSHMLOG\_CLOCK\_ID**

Value: A number from 0 to max of clocks getter in atrshmlog\_get\_clocktime.

Define: ATRSHMLOG\_CLOCK\_ID\_SUFFIX

Rational:

You can get one of the clock functions to deliver the time. If you try others it can be they are not working on the system in place – worse can be an illegal instruction trap.

So if you need a better clock getter its possible to check this in a test program.

For the module the simplest getter is used normally, but you can change this with a define if you have found a better.

## **ATRSHMLOG\_CHECKSUM**

Value: A flag 0 or 1.

Define: ATRSHMLOG\_CHECKSUM\_SUFFIX

Rational:

If you encounter consistency problems it might be needed to switch fences on. But they are also a performance issue. So we try to locate where we need them. To do so you have to check the problem in detail. And for the transfer you will need the checksum then. So you switch the time consuming and eventually misleading checksum calculation on with this.

This should raise the counter for checksum errors in case you need them.

If your consistence problem vanish you have indeed a problem – only the way the checksum was

build now makes a synchronize the hard way – so you then have to switch the fences on and the checksum off. If your problem remains its time for the debugger.

This has to be off in production – solve your problem and dont try to use this a a workaround – then you will never make it to the holy grale of mt users that are worth the mt benefits – you simply exchange low transfer speed for not making your home work ....

## **ATRSHMLOG\_FENCE\_1 to 13**

Value: A flag 0 or 1.

Define : ATRSHMLOG\_FENCE\_1\_SUFFIX TO ATRSHMLOG\_FENCE\_13\_SUFFIX

Rational:

We have in this version no fence active by default. The Intel and AMD systems that are covered simply don't need them as far as I have tested.

If you need fences – or so called memory barriers – then you can switch them on.

To see for the effect can be tricky, so be prepared you have to compile in some check stuff and then can see if it works.

For the fences see the chapter about the things.

## **ATRSHMLOG\_LOGGING\_IS\_OFF\_AT\_START**

Value: A flag 0 or 1.

Define: ATRSHMLOG\_LOGGING\_OFF\_SUFFIX

Rational:

Sometimes you need to start with the log off. So this is the switch to do it. If you don't set it the module will start with logging on.

This is needed in tricky initialization situations when you simply don't know the correct order of execution. It is rare in C, but for C++ this can be very nasty. So if you decide to make the attach somewhere but cannot control otherwise its possible to hold logging till you are ready.

## **ATRSHMLOG\_EVENT\_NULL**

Value : Not relevant: If set it switches on. If not set off. Same for the file : If exists its on, if not exists off.

Define: ATRSHMLOGEVENTNULLSUFFIX

Rational:

You can switch from positive logic for the event initialization to negative logic.

In positive logic the events are set to on – 1 – for default. In negative logic they are set to off -0 .

So you decide if you start with logging events or not at all.

This is only of small help if you not set the events then you need to on – or to off – individual.

To do this you can use the function or the initialize via environment.

## **ATRSHMLOG\_EVENT\_ONOFF**

Value: A vector of event numbers.

Define: ATRSHMLOGEVENTONOFFSUFFIX

Rational:

You can set events different than the default with this. See also ATRSHMLOG\_EVENT\_NULL for this. In positive logic you switch the events off then. In negative logic you set them on.

The vector is in case of the file given by a serious of numbers. Every number is one event index and the number must be within range from 0 to max index.

For the environment we use a C sting with the numbers separated by ':' as in UNIX land. This is true also for mingw. Again a number has to be in range of 0 to max index.

So you can negate the logic, and then switch on only individual events. That's handy if you need only some logging's at all.

The other way you can switch of a group you actually don't need this time. Also handy.

Only one of the things is possible. So you can only do this one time. Switching has no effect after the attach made the event init.

## **ATRSHMLOG\_FETCH\_COUNT**

Value: A number.

Define : -

Rational:

The number of threads the reader c and d start to fetch from the area.

This is per default half the number of buffers. This is in theory the high throughput scenario. In practice it can be reduced to the number of slaves in the programs writing.

Spare CPU this way.

## **ATRSHMLOG\_WRITE\_COUNT**

Value: A number.

Define : -

Rational:

The number of writes the reader c and d start. This is for the number of fetchers a three times high count. So you get a bunch more writes than fetchers.

If you still encounter a blowing reader its time to check if you have a high amount of writes of buffers. Then you must raise the number of writes to keep up with the fetchers and slaves. Depends on the speed of write in last consequence, so I cannot give you a better default than make it three times bigger.

## **ATRSHMLOG\_ALLOC\_ADVANCED**

Value: A number .

Define: -

Rational:

If you have a high amount of logging and you have trouble to keep up with the reader from the beginning on you can set this to allocate additional buffers for the reader in advance to make it till the first writers have done the job. You start in the reader d for now with 1024 buffers. You get 1024 next buffer for an loop with the alloc advanced. Not one, but 1024.

So if you have a thing that needs let say a short lifetime but about 3000 buffers you would need no write, but fetchers and at best 4000 buffers. No overhead and CPU for writers. Fetchers is another story. But enough buffers to catch the log.

Then when the reader stops it writes down all – after your test. So the system can make it without file system io at all (if it isn't for the paging ...).

If you have the need for more buffers keep in mind this is a per 1024 switch or you can run out of memory easily.

Meaning: small numbers are GB ...

## **Functions to use before attach**

The attach is also the initialization of the module. You can set the values for some flags and default values by environment. You can set some of the values alternative before you attach with the functions that follow. For the layer simply check the corresponding thing.

If you use a setter here you have to keep in mind the attach will have the last word for it – it simply ignores if you have a environment too the previous value.

If a thing is initialized more complex you can only change after the attach. This is true for the events. So don't set them before attach. The settings will not make it.

### **atrshmlog\_set\_env\_prefix**

You can set another prefix. This MUST be done before attach – after it simply is not used.

### **atrshmlog\_set\_event\_locks\_max**

You can set the new size of the event flag buffer to use bigger events. Still init is done in attach, so you can not set events in advance.

### **atrshmlog\_set\_buffer\_size**

You can reduce the size for buffers for the dynamically allocated buffers here.

### **atrshmlog\_set\_f\_list\_buffer\_slave\_count**

You can set the number of slaves to start.

### **atrshmlog\_set\_clock\_id**

You can set the id for the clock function to use in the atrshmlog\_get\_clocktime.

### **atrshmlog\_set\_wait\_for\_slaves\_on**

You can set the flag to on.

### **atrshmlog\_set\_wait\_for\_slaves\_off**

You can set the flag to off.

## **atrshmlog\_set\_f\_list\_buffer\_slave\_wait**

You can set the wait time for the slaves .

## **atrshmlog\_set\_prealloc\_buffer\_count**

You can set the number of buffers to alloc in one low level alloc here.

## **atrshmlog\_set\_strategy\_process**

You can set the buffer full strategy default of the process here.

## **atrshmlog\_set\_thread\_fence\_1 to atrshmlog\_set\_thread\_fence\_13**

You can set the flag for use of the fence here.

## **atrshmlog\_set\_init\_buffers\_in\_advance\_on**

You can set the init in advance to on here

## **atrshmlog\_set\_init\_buffers\_in\_advance\_off**

You can set the init in advance to off here.

For other functions you can call them too, but this makes only partly sense to me.

Of course you can call the getter's if you need.

The exception is the get event. This makes only sense after the init which is done in the attach.

## Alphabetical Index

Adjustment.....	209
area...15f., 23ff., 32, 45ff., 49, 68, 72, 80, 85, 100, 102ff., 119, 136, 153, 185, 199, 211, 213ff., 221, 223f., 228ff., 232, 239, 241, 243f., 248ff., 257ff., 269ff., 275, 286, 292, 294f., 298f., 301f., 304ff., 321, 330, 339, 345	
Area.....	307, 313, 339
AREA.....	32
atrshmlog.....	23, 25, 33, 45, 52, 59ff., 66f., 73, 199, 211f., 221f., 242ff., 258, 260ff., 269ff., 291ff., 336f., 339f., 343, 347f.
ATRSHMLOG.....	31, 33f., 40, 53, 70, 87, 96, 101, 161f., 212, 247, 338ff.
atrshmlog_acquire_buffer.....	266
atrshmlog_alloc.....	265
ATRSHMLOG_ALLOC_ADVANCED.....	346
atrshmlog_attach.....	258
ATRSHMLOG_BUFFER_SIZE.....	341
atrshmlog_buffers_preallocoed.....	274
ATRSHMLOG_CHECKSUM.....	343
atrshmlog_cleanup_locks.....	261
ATRSHMLOG_CLOCK_ID.....	343
ATRSHMLOG_COUNT.....	339
atrshmlog_create.....	262
atrshmlog_create_slave.....	271
atrshmlog_decrement_slave_count.....	272
atrshmlog_delete.....	262
ATRSHMLOG_DELIMITER_VALUE.....	340
atrshmlog_dispatch_buffer.....	266
atrshmlog_error_area_count_1.....	306
atrshmlog_error_area_ich_habe_fertig_1.....	306
atrshmlog_error_area_version_1.....	305
atrshmlog_error_attach_1.....	295
atrshmlog_error_attach_2.....	295
atrshmlog_error_attach_3.....	295
atrshmlog_error_attach_4.....	295
atrshmlog_error_attach_5.....	295
atrshmlog_error_attach_6.....	296
atrshmlog_error_attach_7.....	296
atrshmlog_error_buffer_slave_1.....	314
atrshmlog_error_connect_1.....	292
atrshmlog_error_connect_2.....	292
atrshmlog_error_create_1.....	308
atrshmlog_error_create_2.....	308
atrshmlog_error_create_3.....	308
atrshmlog_error_create_4.....	308
atrshmlog_error_error.....	291
atrshmlog_error_error2.....	291
atrshmlog_error_error3.....	292
atrshmlog_error_error4.....	292
atrshmlog_error_error5.....	292
atrshmlog_error_get_autoflush_1.....	315

atrshmlog_error_get_event_1.....	306
atrshmlog_error_get_logging_1.....	307
atrshmlog_error_get_logging_2.....	307
atrshmlog_error_get_logging_3.....	307
atrshmlog_error_get_logging_4.....	307
atrshmlog_error_get_strategy_1.....	314
atrshmlog_error_init_in_write_1.....	296
atrshmlog_error_init_shm_1.....	308
atrshmlog_error_init_shm_2.....	309
atrshmlog_error_init_shm_3.....	309
atrshmlog_error_init_thread_local_1.....	292
atrshmlog_error_mem_to_shm_1.....	293
atrshmlog_error_mem_to_shm_2.....	293
atrshmlog_error_mem_to_shm_3.....	293
atrshmlog_error_mem_to_shm_4.....	293
atrshmlog_error_mem_to_shm_5.....	294
atrshmlog_error_mem_to_shm_6.....	294
atrshmlog_error_mem_to_shm_7.....	294
atrshmlog_error_mem_to_shm_8.....	294
atrshmlog_error_ok.....	291
atrshmlog_error_read_1.....	309
atrshmlog_error_read_2.....	310
atrshmlog_error_read_3.....	310
atrshmlog_error_read_4.....	310
atrshmlog_error_read_5.....	310
atrshmlog_error_read_6.....	311
atrshmlog_error_read_fetch_1.....	311
atrshmlog_error_read_fetch_2.....	311
atrshmlog_error_read_fetch_3.....	312
atrshmlog_error_read_fetch_4.....	312
atrshmlog_error_read_fetch_5.....	312
atrshmlog_error_read_fetch_6.....	312
atrshmlog_error_set_autoflush_1.....	315
atrshmlog_error_set_strategy_1.....	315
atrshmlog_error_verify_1.....	313
atrshmlog_error_verify_2.....	313
atrshmlog_error_verify_3.....	313
atrshmlog_error_verify_4.....	313
atrshmlog_error_verify_5.....	314
atrshmlog_error_verify_6.....	314
atrshmlog_error_write0_1.....	297
atrshmlog_error_write0_10.....	299
atrshmlog_error_write0_2.....	297
atrshmlog_error_write0_3.....	297
atrshmlog_error_write0_4.....	297
atrshmlog_error_write0_5.....	298
atrshmlog_error_write0_6.....	298
atrshmlog_error_write0_7.....	298
atrshmlog_error_write0_8.....	298
atrshmlog_error_write0_9.....	299

atrshmlog_error_write1_1.....	299
atrshmlog_error_write1_10.....	301
atrshmlog_error_write1_11.....	301
atrshmlog_error_write1_12.....	302
atrshmlog_error_write1_13.....	302
atrshmlog_error_write1_2.....	299
atrshmlog_error_write1_3.....	300
atrshmlog_error_write1_4.....	300
atrshmlog_error_write1_5.....	300
atrshmlog_error_write1_6.....	300
atrshmlog_error_write1_7.....	301
atrshmlog_error_write1_8.....	301
atrshmlog_error_write1_9.....	301
atrshmlog_error_write2_1.....	302
atrshmlog_error_write2_10.....	304
atrshmlog_error_write2_11.....	305
atrshmlog_error_write2_12.....	305
atrshmlog_error_write2_13.....	305
atrshmlog_error_write2_2.....	302
atrshmlog_error_write2_3.....	303
atrshmlog_error_write2_4.....	303
atrshmlog_error_write2_5.....	303
atrshmlog_error_write2_6.....	303
atrshmlog_error_write2_7.....	304
atrshmlog_error_write2_8.....	304
atrshmlog_error_write2_9.....	304
ATRSHMLOG_EVENT_COUNT_MAX.....	341
ATRSHMLOG_EVENT_NULL.....	344
ATRSHMLOG_EVENT_ONOFF.....	345
atrshmlog_event_t.....	247
atrshmlog_exit_cleanup.....	271
atrshmlog_f_list_buffer_slave_proc.....	272
ATRSHMLOG_FENCE_1.....	344
ATRSHMLOG_FETCH_COUNT.....	345
atrshmlog_flush.....	267
atrshmlog_free.....	267
atrshmlog_get_acquire_count.....	277
atrshmlog_get_area.....	262
atrshmlog_get_area_count.....	262
atrshmlog_get_area_ich_habe_fertig.....	263
atrshmlog_get_area_version.....	263
atrshmlog_get_autoflush_process.....	277
atrshmlog_get_buffer_id.....	277
atrshmlog_get_buffer_max_size.....	277
atrshmlog_get_buffer_size.....	277
atrshmlog_get_checksum.....	277
atrshmlog_get_clock_id.....	277
atrshmlog_get_env.....	273
atrshmlog_get_env_id_suffix.....	274, 277
atrshmlog_get_env_prefix.....	274

atrshmlog_get_env_shmid.....	274
atrshmlog_get_event.....	278
atrshmlog_get_event_locks_max.....	278
atrshmlog_get_f_list_buffer_slave_count.....	279
atrshmlog_get_f_list_buffer_slave_wait.....	279
atrshmlog_get_init_buffers_in_advance.....	279
atrshmlog_get_inittime.....	279
atrshmlog_get_inittime_tsc_after.....	279
atrshmlog_get_inittime_tsc_before.....	279
atrshmlog_get_logging.....	275
atrshmlog_get_minor_version.....	279
atrshmlog_get_next_slave_local.....	273
atrshmlog_get_patch_version.....	279
atrshmlog_get_prealloc_buffer_count.....	279
atrshmlog_get_realtime.....	276
atrshmlog_get_shmid.....	279
atrshmlog_get_statistics.....	276
atrshmlog_get_statistics_max_index.....	279
atrshmlog_get_strategy.....	279
atrshmlog_get_strategy_process.....	279
atrshmlog_get_thread_fence_1.....	278
atrshmlog_get_thread_fence_10.....	278
atrshmlog_get_thread_fence_11.....	278
atrshmlog_get_thread_fence_12.....	278
atrshmlog_get_thread_fence_13.....	279
atrshmlog_get_thread_fence_2.....	278
atrshmlog_get_thread_fence_3.....	278
atrshmlog_get_thread_fence_4.....	278
atrshmlog_get_thread_fence_5.....	278
atrshmlog_get_thread_fence_6.....	278
atrshmlog_get_thread_fence_7.....	278
atrshmlog_get_thread_fence_8.....	278
atrshmlog_get_thread_fence_9.....	278
atrshmlog_get_thread_local_tid.....	280
atrshmlog_get_tid.....	280
atrshmlog_get_version.....	280
atrshmlog_get_wait_for_slaves.....	280
ATRSHMLOG_ID.....	339
atrshmlog_il_connect_buffers_list.....	266
atrshmlog_il_get_raw_buffers.....	275
atrshmlog_init_events.....	277
ATRSHMLOG_INIT_IN_ADVANCE.....	340
atrshmlog_init_in_write.....	269
atrshmlog_init_shm_log.....	260
atrshmlog_init_thread_local.....	269
atrshmlog_init_via_env.....	282
atrshmlog_init_via_file.....	282
atrshmlog_int32_t.....	246
atrshmlog_internal_time_t.....	246
atrshmlog_internal.h.....	25, 61, 222, 248

ATRSHMLOG_LOGGING_IS_OFF_AT_START.....	344
atrshmlog_perlwrapper.c.....	125f.
atrshmlog_pid_t.....	246
ATRSHMLOG_PREALLOC_COUNT.....	341
atrshmlog_read.....	265
atrshmlog_read_fetch.....	264
atrshmlog_remove_slave_via_local.....	273
atrshmlog_ret_t.....	247
atrshmlog_reuse_thread_buffers.....	270
atrshmlog_set_area_ich_habe_fertig.....	263
atrshmlog_set_autoflush.....	280
atrshmlog_set_autoflush_process.....	280
atrshmlog_set_buffer_size.....	280
atrshmlog_set_checksum.....	280
atrshmlog_set_clock_id.....	280
atrshmlog_set_env_prefix.....	274
atrshmlog_set_event.....	280
atrshmlog_set_event_locks_max.....	276
atrshmlog_set_f_list_buffer_slave_count.....	281
atrshmlog_set_f_list_buffer_slave_run_off.....	282
atrshmlog_set_f_list_buffer_slave_wait.....	282
atrshmlog_set_init_buffers_in_advance_off.....	280
atrshmlog_set_init_buffers_in_advance_on.....	280
atrshmlog_set_logging_process_off.....	281
atrshmlog_set_logging_process_off_final.....	281
atrshmlog_set_logging_process_on.....	282
atrshmlog_set_prealloc_buffer_count.....	282
atrshmlog_set_strategy.....	282
atrshmlog_set_strategy_process.....	282
atrshmlog_set_thread_fence.....	282
atrshmlog_set_thread_fence_1.....	280
atrshmlog_set_thread_fence_10.....	281
atrshmlog_set_thread_fence_11.....	281
atrshmlog_set_thread_fence_12.....	281
atrshmlog_set_thread_fence_13.....	281
atrshmlog_set_thread_fence_2.....	280
atrshmlog_set_thread_fence_3.....	281
atrshmlog_set_thread_fence_4.....	281
atrshmlog_set_thread_fence_5.....	281
atrshmlog_set_thread_fence_6.....	281
atrshmlog_set_thread_fence_7.....	281
atrshmlog_set_thread_fence_8.....	281
atrshmlog_set_thread_fence_9.....	281
atrshmlog_set_wait_for_slaves_off.....	282
atrshmlog_set_wait_for_slaves_on.....	282
ATRSHMLOG_SLAVE_COUNT.....	342
ATRSHMLOG_SLAVE_WAIT_NANOS.....	342
atrshmlog_sleep_nanos.....	276
atrshmlog_stop.....	270
ATRSHMLOG_STRATEGY.....	340

atrshmlog_strategy_adaptive.....	337
atrshmlog_strategy_adaptive_fast.....	337
atrshmlog_strategy_adaptive_very_fast.....	337
atrshmlog_strategy_discard.....	336
atrshmlog_strategy_spin_loop.....	336
atrshmlog_strategy_wait.....	59, 336
ATRSHMLOG_STRATEGY_WAIT.....	340
ATRSHMLOG_STRATEGY_WAIT_TIME.....	340
atrshmlog_swigwrapper.c.....	142f.
atrshmlog_tid_t.....	246
atrshmlog_time_nanoseconds_t.....	246
atrshmlog_time_seconds_t.....	246
atrshmlog_time_t.....	246
atrshmlog_transfer_mem_to_shm.....	263
atrshmlog_turn_logging_off.....	270
atrshmlog_turn_slave_off.....	273
atrshmlog_verify.....	261
ATRSHMLOG_WAIT_FOR_SLAVES_ON.....	342
ATRSHMLOG_WRITE_COUNT.....	346
atrshmlog_write0.....	267
atrshmlog_write1.....	267
atrshmlog_write2.....	267
atrshmlog.h.....	23, 25, 31, 33f., 40, 45, 52f., 59ff., 66f., 70, 73, 87, 96, 101, 161f., 199, 211f., 221f., 242ff., 258, 260ff., 266f., 269ff., 291ff., 336ff.
Atrshmlog.i.....	124
ATRSHMLOG.java.....	87, 96, 101
atrshmlogcalc.....	23
atrshmlogcheckcomplete.....	23, 38
Atrshmlogcheckcomplete.....	28
atrshmlogchecksystem.....	23, 40, 53
Atrshmlogchecksystem.....	28
atrshmlogconv.....	23, 49, 189
atrshmlogconv.....	50
atrshmlogconvert.....	23
atrshmlogcreate.....	23, 44, 100, 102, 119, 136, 153, 184, 197
atrshmlogdefect.....	24, 209
atrshmlogdump.....	24
atrshmlogerror.....	24
atrshmlogfinish.....	24
atrshmloginit.....	24, 45, 102
atrshmlogjnipackage.c.....	23, 31, 86f.
atrshmlogmodule.c.....	109f.
atrshmlogoff.....	24, 307
atrshmlogon.....	24
atrshmlogreaderd.....	24f., 47, 210, 212
atrshmlogreset.....	24
atrshmlogsignalreader.....	24, 202
atrshmlogsignalwriter.....	24
atrshmlogsort.....	24
atrshmlogstat.....	24, 103, 218, 316

atrshmlogstopreader.....	25, 47, 188, 202
atrshmlogtest.pl.....	133
atrshmlogtest.py.....	116
atrshmlogtest.tcl.....	150
atrshmlogtest00.....	25, 46, 186, 200
atrshmlogtest01.....	25
atrshmlogtest03.....	195, 209
atrshmlogverify.....	25
basedir.....	161f., 192
BASEDIR.....	22, 26, 30, 35, 41, 43, 81f., 88, 95, 111, 116, 127, 133, 144, 150, 161f., 178, 205, 245
buddydoc.sh.....	27
buddylib.sh.....	25, 26
centos.....	161
CentOS.....	56, 159, 164
Cleanall.sh.....	28
compile_jni_stub.sh.....	31, 89, 94
compile_perl_stub.sh.....	128, 132
compile_python_stub.sh.....	112, 115
compile_swig_stub.sh.....	145, 149
compile_to_class_package_version.sh.....	31, 89, 94
create_header_package_version.sh.....	31, 89f., 95
create_jni_lib.sh.....	31, 90, 95, 97, 99, 205
create_perl_lib.sh.....	128, 132, 135
create_python_lib.sh.....	112, 115, 118
create_swig_lib.sh.....	145, 149, 152
cygwin.....	22, 26f., 30, 162, 176ff., 190f., 196ff., 204f., 245, 249, 251, 262, 275f., 334, 339
Cygwin.....	176, 240
dot.java.sh.....	30f., 90, 95, 97f., 205
dot.perl.sh.....	132, 134
dot.platform.sh.....	26
dot.python.sh.....	115, 117
dot.swig.sh.....	149, 151
ell.sh.....	26ff., 161
environment.....	21, 26, 29ff., 33f., 41, 44f., 63, 65, 68, 80, 82, 86, 90, 95, 98, 101ff., 107, 109, 115, 117, 125, 132, 134, 142, 149, 151, 176, 184, 191f., 205, 215, 221, 225, 239, 249f., 259, 273ff., 277, 295, 297, 300, 303, 306, 309, 313, 329f., 333, 338f., 345, 347
Environment.....	338
Error codes.....	291
event.33, 60, 70, 73f., 76, 82, 103f., 247, 260, 276ff., 280, 297, 300, 303, 306, 327f., 333, 339, 341, 345, 347f.	
EVENT.....	277, 341, 344f.
fences.....	241
fenster;plural.....	15, 22, 27f., 63, 88, 103, 105f., 176, 191, 197, 209, 240f., 245, 251, 276
g++14w.sh.....	26ff., 162
g99.sh.....	26ff., 161
getfrommain.sh.....	90f., 97, 112f., 128ff., 145ff., 205
java.....	18, 20, 23, 29ff., 35, 62, 86ff., 95ff., 101f., 106, 204f., 218, 231, 239f., 249
Java.....	31, 230
JAVA.....	30, 97
jni.....	23, 29ff., 62, 86f., 89f., 94ff., 99ff., 204ff., 239

JNI.....	30
korn.....	14, 17, 20, 80f.

