# SrlApi

Srl API is your gateway to software resources.

## Logging

Log API let you define message levels. Levels allow you to let your debug code in the source, and only compile it when needed.

In order, levels are:

- NONE
- TRACE
- DEBUG
- MAX

When writing your software, you decide what level the message is for. When compiling or running you software, you decide what minimal level your code must have to be printed.

- `srl_log(level, "message")` prints a message
- `srl_log_printf(level, "message_with_format", arguments...)` prints a printf-like message

Arguments in printf-like version may be not evaluated if level is not sufficient. Therefore you **MUST NOT** put expressions with side effects in the parameter list. ie do **not** do this:

```
srl_log_printf(DEBUG, "i=%d\n", i++);
```

## Mwmr fifos

- `srl_mwmr_read(fifo, buffer, size)` reads `size` 32-bit words from fifo to buffer. `size` must be a multiple of fifo width.
- `srl_mwmr_write(fifo, buffer, size)` writes `size` 32-bit words from buffer to fifo. `size` must be a multiple of fifo width.

## Locks

- `srl_lock_lock( lock )` takes the lock, waiting if necessary
- `srl_lock_unlock( lock )` releases the lock

## Barriers

- `srl_barrier_wait( barrier )` waits for a barrier-global synchronization

## Other APIs

- `srl_run_cycles( N )` tells the simulation environment the simulation should run at least N cycles while in this call. This makes sense only for virtually sunthetised tasks, otherwise, this call is a noop.
- `srl_mwmr_config( controller_name, reg_n, value )` puts value `value` in config register `reg_n` of specified controller

- `srl_mwmr_status( controller_name, reg_n )` reads status register `reg_n` of specified controller, returns a int32_t
- `srl_assert( cond )` checks `cond` is true, fatally fails otherwise