mµOS - My micro OS

Contents

a small operating system for microcontrollers

Chapter 1

Introduction

1.1 Concept and Goals

mµOS is supposed to be a general purpose Operating-System for small Microcontrollers. With proper configuration it scales down to be useable on AVR ATtiny MPUs. The main focus is on getting Projects efficently and robustly done in short time.

The core of mµOS is a Scheduler which serves function calls from a few queues (Timer and 2 Priorities by default). This approach was choosen to minimize resource demands. Everything is served only by one Stack. There are no threads or context switches, but there are some facilities to call the Scheduler recursively.

Users code always runs synchronously from these queues which are scheduled by the mainloop provided by mµOS. Interrupts are handled by *drivers* and do the bare minimum necessary to serve the request, postpone any further processing themself onto the work queues.

This means that there are less possiblities of race conditions and in many cases a user doesn't need to care for synchronization (unless writing drivers).

Still any resource which is not managed by mµOS is available to the user and can be freely accessed and configured. This is a deliberate design decision to put mµOS between bare-metal programming and some more complete OS.

Whenever reasonable mµOS reuses existing Platform and C libraries, but also adds its own concepts and API's breaking history with POSIX and other common API's whenever there are some reasons to do so.

1.2 Features

- Most things in mµOS are configurable in very small detail. The build system only links parts which are referenced into the resulting firmware, thus it hard to tell what size mµOS has, because it depends on the application build on top. When slimmed down then sizes smaller than 1kByte and RAM usage of only few bytes are possible.
- The mainloop pops functions from work queues, executing them and puts the µC to sleep when nothing needs to be done. Any Interrupt wakes the mainloop, the Interupt handler may put work onto the queues which in turn get called as soon the interrupt handler finishes. Scheduling latency is typically 20µs on a 8MHz AVR with the code yet unoptimized.
- *muos_yield()* and *muos_wait()* calls can be used to suspend the current job and call the mainloop recursively.

- mµOS uses one Hardware counter as main-timer, any timer, 8 or 16 bit can be used for this. The prescaler of this timer is configureable to give a lot freedom on the desired clock granularity and interrupt load. The overflow interrupt is used to increment a global counter and one of the output compare units is used to schedule wakeups of the mainloop.
- There are (configureable) queues:
  - *clpq* the timer queue, using a sliding window priority-queue implementation to schedule work within near future, attainable timespans depend on the timer prescaler and *shortclock* type. *clpq* jobs are executed with the highest priority.
  - *hpq* and *bgq* are simple queues for high and low priority jobs. The scheduler executes from the front of this queues, first all jobs in the *hpq* and then all jobs in the *bgq*. One can push new jobs to then end or to the front of these queues, giving roughly 4 different priority levels. One should split work to be done into reasonable small parts and queue them to either work queue. Drivers are also supposed to do most of the work outside of Interrupts by pushing them to the queues.
- Errors can happen asynchronously in interrupt handlers. Any error has a correspondending bit in a bit-array which flags its occurence. When any errors happened in interrupt handlers are not handled there, then the mainloop first calls a user-defined error handling function.

1.3 Supported Hardware

Hardware support is added on demand. The following Platforms are tested and known to work. But not all features are supported on all microcontrollers. Other ones may work and can be added with little effort.

Future plans include to port mµOS to STM32 and other platforms.

1.4 AVR

attiny84
: Bare chip, programmed with usbasp/avrdude.

attiny85
: Digispark (http://digistump.com/products/1) with micronucleus bootloader.

atmega328p
: Arduino Nano. This is the reference Platform. Most things are tested and developed here.

Chapter 2

Infrastructure

2.1 Requirements

mµOS is developed on Linux with GNU-Tools. As far these are available it should be compilable on other Operating Systems as well.

When versions are noted, this are the versions which are proven to work, if for some reason only older versions are available and work for you, please report this. The exact package names may differ depending on the Distribution.

2.1.1 For building Projects

make (>= 4.0)
GNU make

gcc-avr (>= 4.7)
GNU C compiler, version 4.7 is needed at least for *__flash* memory support. MµOS relies on -flto which will improve with newer Compiler versions.

avr-libc (>= 1.8)
The GNU libc for AVR processors

binutils-avr
Tools for building

2.1.2 For flashing

avrdude (>= 6.1)
Needed for flashing via USBasp or Arduino bootloaders (and possibly some more in future)

micronucleus (>= 2.0)
Needed for flashing micronucleus based bootloaders (DigiSpark and other attiny based products)

2.1.3 Revision control

git
For cloning and maintaining the mµOS source tree

2.1.4 Building the Documentation

lua
used by the *pipadoc* documentation extractor (included in mµOS)

asciidoc
for generating HTML

a2x, dblatex, ...
For generating PDF documentation

2.2 Development

MµOS are developed for the avr-gcc/avr-libc toolchain, on 8-bit AVR's (ATTiny, ATMega) with some portability in mind. It is later planned to extend this to other platforms like 32bit ARM (STM32). Direction is still to stay as simple as the avr toolchain is to make mirgration for any programmer coming from Atmel programming to STM32 (and perhaps others) as easy as possible. Still we will make no compromises in functionality where required.

As Free Software project, mµOS is open to any contributor. Development will progress on demand. Whatever is needed will be integrated.

mµOS comes comes with a Makefile managing all project tasks. For the example Project bundled with the mµOS source, everything is configured inside this Makefile (this may be refined in future).

2.2.1 Build System

The build-system tries to do exact dependency tracking and building things in parallel. The Makefile in the users *src/* directory includes the main *muos/muos.mk* Makefile, which in turn includes sub Makefiles for to support different flashing tools and different platforms.

By default the make output is silent. If one adds the "MAKE_DEBUG=1" flag to the make invocation then parallel builds are disabled and the it produces verbose output.

The most importnat build targets are:

all
Builds everything

upload
Flashes the firmware to the target

doc
Build the documentation

issues
Extracts `muos_issues.txt` and builds `muos_issues.html`

clean
Deletes anything that got build

2.2.2 Documenation

The Documentation is writiten in *asciidoc* and distributed in serveral `.txt` files and added as special comments to the sources. The *pipadoc* tool shipped with mµOS is used to extract this Documentation into single text files. For building the Documentation one needs a *lua* interpreter, the *asciidoc* toolchain and perhaps *dblatex* for pdf generation.

2.2.3 Issues

Issues are tagged inside source and documentation files. *pipadoc* takes care of generating a `muos_issues.txt` and `muos_issues.html`. There are deliberately only 3 tags for issues:

`FIXME`
: Known bug/problem which leads to failures and must be fixed.

`TODO`
: Intended functionality which is not yet implemented/finished. If not used this should not run into errorneous behaviour.

`PLANNED`
: Notes about future features and ideas, may be subject to refinement and should not affect the existing functionality at all.

The build system also prints these issues lists on stderr, so that the relevant source positions are navigateable from an IDE. Which issues are presented there is decided on the currently checked out git branch.

In *master* only `FIXME` are shown. In *devel* `FIXME` and `TODO` are shown. for any other branch `FIXME`, `TODO` are shown and `PLANNED` are filtered by that only issues where the pathname matches the current branchname are shown.

Chapter 3

MµOS

3.1 The Mainloop

In default configuration mµOS provides a mainloop which schedules queues holding user-defined functions to be called. *main()* is part of mµOS and not defined in user programs.

At start up mµOS first schedules a (configurable) *init()* function which does the initialization defined by the user. This init is called with the timer and drivers still uninitialized/stopped, second after the *init()* function *muos_start()* is scheduled which then sets up all drivers and starts the timer. Upon its completion everything is initialized and the system runs.

Some drivers provide configurable hooks which are scheduled when some event happened (examples are like lineedit completed with the user pressing `[RETURN]`)

For every iteration the mainloop sets a global variable to the current time which then can be queried with *muos_now()*. This time guaranteed to stay stable for the whole mainloop iteration with only the exceptions that nested mainloop invocation (*muos_wait()* and *muos_yield()* will update it). Using this should be the preferred way to get the current time over the direct clock api's because it is faster to access and provides the necessary stability as time tag for this iteration.

If there are any pending errors left from asynchronous calls (Interrupts and Drivers) it calls a user defined error handler. This is done at highest priority with interrupts still disabled. The error-handler is called only once per mainloop iteration, even if it leaves errors pending.

Then the mainloop checks if there is any work on the queues (see below) and executes these functions appropriately. When there is nothing left to do it either puts the µC into sleep or, for very short times, does a busy loop to ensure exact timing.

3.2 Scheduler Queues

By default there are 3 queues served by the mainloop. Each of them can be configured in some details and disabled if not required.

3.2.1 clock priority queue

Uses the clock to wake the mainloop for scheduling events in *near future*. *near future* is defined by the clock configuration, times longer than the *muos_shortclock* type can not be handled. This times are also affected by the timers prescaler, the faster the clock runs, the shorter are the time spans it can handle. Common ranges are from few milliseconds to many hours.

Events scheduled by the clpq have the highest priority, but are still called synchronously within the mainloop and interrupts enabled.

3.2.1.1 clpq API

The priority queue uses a sliding-window implementation, this allows to add times covering the while range of the *muos_shortclock* datatype.

The *clpq* uses the priority-queue implementation (and thus the types) from

lib/spriq.

Schedule a function at the given time

```
void muos_clpq_at (
       muos_spriq_priority base,
       muos_spriq_priority when,
       muos_spriq_function what
)
```

base
time used as relative base for the timing calculation

when
offset to base for the destination time

what
function to be scheduled

For scheduling on time `base` has some more contraints to handle overflows correctly. The *clpq* handles that, refer to the source for details.

Reschedule an event

```
void muos_repeat (
       const struct muos_spriq_entry* event,
       muos_spriq_priority when
)
```

event
the event pointer passed which get passed into queued functions

when
time when to repeat this event

3.2.1.2 clpq configuration

Only the length is configured for the *clpq*, most of its configurations come from the clock and spriq configuration see below.

MUOS_CLPQ_LENGTH
Number of entries the scheduling Queue can hold, set to 0 to disable the clpq

3.2.2 Work Queues

The mainloop will schedule work on the Workqueues until nothing left to be done. Work is always popped from the front of these queues. There are API's to push work on the back (preferred case) and on the front of this queues. Pushing something on the front of a queue means that it will be scheduled immediately before anything else pending. Thus one get 4 priority levels for work from this queues.

One can push an optional *intptr_t* argument together with each scheduled function to give a limited ability to pass data around. If this argument is used it will use additional memory in the queue.

The underlying queue implementation in [?] can be configured in several ways.

3.2.3 high priority queue

Scheduled with a priority below the clpq and above bgq.

3.2.3.1 hpq API

Schedule functions at high priority

```
muos_error muos_hpq_pushback (muos_queue_function fn)
muos_error muos_hpq_pushback_arg (muos_queue_function_arg fn, intptr_t arg)
muos_error muos_hpq_pushfront (muos_queue_function fn)
muos_error muos_hpq_pushfront_arg (muos_queue_function_arg fn, intptr_t arg)
muos_error muos_hpq_pushback_isr (muos_queue_function fn, bool schedule)
muos_error muos_hpq_pushback_arg_isr (muos_queue_function_arg fn, intptr_t arg, bool  ↩
    schedule)
muos_error muos_hpq_pushfront_isr (muos_queue_function fn, bool schedule)
muos_error muos_hpq_pushfront_arg_isr (muos_queue_function_arg fn, intptr_t arg, bool  ↩
    schedule)
```

fn
: function to schedule

arg
: argument to pass to the function

schedule
: when false the scheduler will not check for functions scheduled after a wake up from sleep.

The *_isr variants of these functions are intended to be called from Interrupt handlers or contexts where interrupts are already disabled.

These functions return *muos_success* on success and *muos_error_hpq_overflow* on error.

```
intptr_t muos_hpq_pop_isr (void)
```

removes and returns the first element (argument) from the hpq. Must be called while interrupts are still disabled at the start of scheduled functions. (Note: no safety net when the caller didn't push an argument)

3.2.3.2 hpq configuration

MUOS_HPQ_LENGTH
: How many entries the high priority queue can hold, set to 0 to disable the hpq

3.2.4 background queue

Scheduled with the lowest priority.

3.2.4.1 bgq API

Schedule functions at background priority

```
muos_error muos_bgq_pushback (muos_queue_function fn)
muos_error muos_bgq_pushback_arg (muos_queue_function_arg fn, intptr_t arg)
muos_error muos_bgq_pushfront (muos_queue_function fn)
muos_error muos_bgq_pushfront_arg (muos_queue_function_arg fn, intptr_t arg)
muos_error muos_bgq_pushback_isr (muos_queue_function fn, bool schedule)
muos_error muos_bgq_pushback_arg_isr (muos_queue_function_arg fn, intptr_t arg, bool  ↩
    schedule)
muos_error muos_bgq_pushfront_isr (muos_queue_function fn, bool schedule)
muos_error muos_bgq_pushfront_arg_isr (muos_queue_function_arg fn, intptr_t arg, bool  ↩
    schedule)
```

fn
: function to schedule

arg
: argument to pass to the function

schedule
: when false the scheduler will not check for functions scheduled after a wake up from sleep.

The *_isr variants of these functions are intended to be called from Interrupt handlers or contexts where interrupts are already disabled.

These functions return *muos_success* on success and *muos_error_bgq_overflow* on error.

```
intptr_t muos_bgq_pop_isr (void)
```

removes and returns the first element (argument) from the bgq. Must be called while interrupts are still disabled at the start of scheduled functions. (Note: no safety net when the caller didn't push an argument)

3.2.4.2 bgq configuration

MUOS_BGQ_LENGTH
: How many entries the background queue can hold, set to 0 to disable the bgq

3.2.5 Mainloop Control

3.2.5.1 API

Wait for some condition come true

```
typedef bool (*muos_wait_fn)(intptr_t)

muos_error muos_wait (muos_wait_fn fn, intptr_t param, muos_shortclock timeout)
```

fn
: function checking for some condition, must return *false* while the condition is not met and finally *true* on success. Can be NULL, then *muos_wait()* uses only the timeout for the wait.

param
: an optinal intptr_t argument passed to the test function

timeout
: time limit for the wait

Calls a recursive mainloop and with testing for a given condition for some time.

Care must be taken that I/O (and other things) are not anymore in order when the mainloop is called recursively. Often it is better to avoid waiting and divide the work into smaller tasks which are put in order on the work queues.

Because of stack limits entering the mainloop recursively is limited. One should always expect that a wait can return instantly with *muos_warn_sched_depth*.

muos_wait() is only available when MUOS_SCHED_DEPTH is defined.

Returns

muos_success
the wait condition got met

muos_warn_sched_depth
depth limit for recursive mainloops hit

muos_warn_wait_timeout
timed out

Enter Mainloop recursively, scheduling other jobs

```
muos_error muos_yield (uint8_t count)
```

count
number of jobs to schedule, must be less than 254

Calls a recursive mainloop executing at most *count* jobs. A job here is any one thing queued on any of the works queues. Returns if nothing left to do the *count* limit got reached or the scheduler depth limit got reached. Same precautions as on *muos_wait()* apply.

Yielding is applicable when one has code which loops for some extended time but shall not stall the work to be done **and** this code will never be called recursively.

muos_yield() is only available when MUOS_SCHED_DEPTH is defined.

Returns

muos_warn_sched_depth
depth limit for recursive mainloops hit

muos_success
yielded at most *count* times

3.2.5.2 configuration

MUOS_INITFN
Name of the user-defined initialization function. This function is pushed on the hpq (or, if not available the bgq) when starting up. It is responsible for initializing the system. Interrupts are still disabled and the clock is stopped and will be started after this init function returns.

MUOS_SCHED_DEPTH
Set the maximum depth for nested scheduler calls (muos_wait() muos_yield()). When this depth is exceeded, these calls return immediately, flagging *muos_warn_sched_depth*. When not defined then the wait and yield functions become unavailable

MUOS_SCHED_SLEEP
Set sleep mode for the main loop if supported

MUOS_YIELD_DEPTH
Set the maximum depth for scheduler muos_yield() calls. When this depth is exceeded, the calls return immediately with *muos_warn_sched_depth*. When not defined then the yield function becomes unavailable.

3.3 The Clock

The *clock* is one of the most important parts of the OS. Many core parts and other drivers rely on it.

Only one hardware timer is used mµOS to implement the global clock. This can be any timer (8bit, 16bit). MµOS uses its overflow interrupt and needs one compare-match unit timer for scheduling wake ups. The configuration of this timer (prescaler, which hardware timer to be used) is fully customizeable to the user.

The clock is started at the begin of the operations (after the users *init* job got called) and then left freely running.

The clock API also holds a function to calibrate the µC main frequency with some external signal, as far this is supported by the hardware.

Datatypes The full time is defined by the global overflow counter and the timers internal hardware register. This overflow counter can be selected from various unsigned integer types (16, 32, 64 bit wide), the internal timer counter register extends this value by its size, enabling rather high resolution clocks from 24 up to 80 bits precision. Depending on the circumstances one should select a proper size so that overflows don't happen or do not matter.

There are following types used:

`muos_clock`
: Generic type used for moderate long time spans. Depending on configuration it overflows rarely (or preferably never). This is the primary datatype for the clock and the *global overflow counter*

`muos_shortclock`
: Type for short time intervals. Used in the clock priority for events which are scheduled within shorter times.

`muos_hwclock`
: The type of the hardware timer. Usually 8 or 16 bits wide

`muos_fullclock`
: a structure containing the whole clock state with the high bits stored as muos_clock and the low bits stored as muos_hwclock. When properly configured this state should never overflow for the application run time.

Some conclusions about the Datatypes

- The default for the overflow counter is *uint32_t*, calculations show that 16bit makes hardly any sense, because it would overflow quite often even in slow increment configurations.
- More than 32 bit is only needed for fast running clocks or when very long run times are intended.
- Choosing 16 bit hwclock when it is available will have less interrupt load but needs more memory. When in doubt, it is not mandatory when the timer prescaler is not on the fastest setting. This is especially useful if only one 16 bit timer is available and is needed for something else. However some mµOS drivers may require a 16 bit timer.
- *muos_clock* alone overflows quite often but using a 64 bit datatype for it would need a lot memory in the queues (and 64 bit math libs).

3.3.1 API

Time calculation macros

```
MUOS_CLOCK_SECONDS(t)
MUOS_CLOCK_MILLISECONDS(t)
MUOS_CLOCK_MICROSECONDS(t)
```

t
time to be converted

The clock runs on timer ticks. These macros convert time from second, ms, µs to ticks. Because this needs to result in a integer number of ticks, the result might be inexact and add a slight drift.

The correct configuration of F_CPU is mandatory for these macros to work correctly.

The fullclock datatype

```
typedef struct {
 muos_clock coarse;
 muos_hwclock fine;
} muos_fullclock;
```

coarse
the global counter for hardware overflows

fine
the hardware part of the clock

This is the type with the widest range. With sane configurations it can be ensured that *muos_fullclock* never overflows. The drawback is that it needs the most memory for its representaton which becomes a problem when storing multiple timestamps in queues. Most often the smaller datatypes *muos_clock* and *muos_shortclock* are more approbiate when one handles overflows correctly.

Event time

```
muos_clock muos_now (void)
```

This function is very cheap and gives a consistent time throughout the whole mainloop iteration.

Returns time if each mainloop (also recursive from *muos_wait()* and *muos_yield()*) iteration start.

Query Time

```
muos_clock muos_clock_now (void)
muos_clock muos_clock_now_isr (void)
```

Returns the current time as queried from the hardware.

```
muos_shortclock muos_clock_shortnow (void)
muos_shortclock muos_clock_shortnow_isr (void)
```

Returns the current time as queried from the hardware, using the *muos_shortclock* datatype.

```
muos_fullclock muos_clock_fullnow (void)
muos_fullclock muos_clock_fullnow_isr (void)
```

Returns the current time as queried from the hardware, using the *muos_fullclock* datatype.

Time difference between two timestamps

```
muos_clock muos_clock_elapsed (muos_clock now, muos_clock start)
```

now
End of the timespan to be calculated

start
Begin of the timespan to to be calculated

returns the time difference between *now* and *start*. The result is always positive or zero. Simple overflows on the arguments are respected. Thus the full range of *muos_clock* is available.

Calibrate the µC Frequency

```
void muos_clock_calibrate (const muos_clock now, const muos_clock sync)
```

now
Time at which the external sync signal happened

sync
Timespan which should be elapsed since the last call of this function

For µC's which support it, the main frequency can be calibrated by some external signal. For example with some 1 second pulse from a RTC one could call `muos_clock_calibrate (muos_clock_now(), MUOS_CLOCK_SECONDS(1))` upon receiving this signal.

There is a special case that if *sync* is 0 then only the internal state is recorded but no frequency calibration is executed. This is to be used for initialization or when the time elapsed since the last call can't be determined.

Note that frequency calibration is often too coarse to archive perfect synchronization. You should expect some drift remaining. The configuration specially includes a deadband for this. Otherwise when calibration tries to constantly change the frequency there would be very much jitter on the timing.

3.3.2 configuration

MUOS_CLOCK_CALIBRATE
When defined the frequency calibration API for syncing the µC frequency clock with some external signal is enabled.

MUOS_CLOCK_CALIBRATE_DEADBAND
Adds a deadband/range around the target clock. On µC where the clock calibration is rather coarse an exact match can not be reached and constant readjustments will result in high jitter.

MUOS_CLOCK_CALIBRATE_MAX_DEVIATION
When the time elapsed since the last sync differs more than +/- this value, the calibration measurement is reset and starts over. This filters out glitches from API calls.

MUOS_CLOCK_HW
The Hardware timer to use. This is a hardware dependent config. Timer hardware setup is tightly bound to the hardware capabilities check for the respective hardware documentation about possible settings.

MUOS_CLOCK_HW_PRESCALER
Prescaler from some hardware defined master clock. Possible Values depend on the hardware.

MUOS_CLOCK_SHORT_TYPE
The type used for short time spans. Should be some unsigned integer. What suits best depends on the application, often 16 bit is enough with higher Prescaler.

MUOS_CLOCK_TYPE
The type used for the primary clock functions. Should be some unsigned integer, 32 bit width is recommended.

3.4 Error handling

Errors can happen synchronously by returning a error code directly from the called function or asynchronously by setting a flag from interrupts and drivers. This two approaches are necessary because errors can not always be handled where they happen.

The synchronous way is quite straightforward. If a function does return anything but *muos_success* some error happened and can be handled right away.

Asynchronous errors are different as they can be flagged at any time (from interrupts) which can not handle them appropriately. There is a flag (bitfield) for any possible errorcode, but these don't queue up. This means asynchronous errors need to be handled as soon as possible.

For to do this mµOS supports a global error handling function which is called in the main-loop when any error is pending. This error handling function has the highest priority and upon enter, interrupts are still disabled. It is advised to enable interrupts there as soon as possible (there is no need to disabled them again, the main-loop takes care of that). The global error handling function only needs to handle most important failures. Any errors not handled there can be handled elsewhere from normal queued functions as well.

Errors can be flagged and checked, checking for an error automatically clears the flag. There is also a function to check if an error flag is set without clearing it.

3.4.1 Error codes

The following error codes are used nby mµOS, depending on configuration, some might be disabled when the respective subsystem/driver is not enabled.

muos_error_error
: unspecified error

muos_warn_sched_depth
: recursive scheduler calls exceeded

muos_warn_wait_timeout
: muos_wait() timed out

muos_error_clpq_overflow
: clock priority queue full

muos_error_hpq_overflow
: high priority queue full

muos_error_bgq_overflow
: background priority queue full

muos_error_nodev
: device index out of range

muos_error_sm_state
: state transition not possible

muos_error_tx_blocked
: tx is blocked by another job

muos_error_tx_buffer_overflow
: To much data to send

muos_error_rx_blocked
: rx is blocked by another job

muos_error_rx_buffer_overflow
dropped received data (user code)

muos_error_rx_buffer_underflow
read while no data available

muos_error_rx_frame
wrong stop bit timing

muos_error_rx_overrun
dropped received data (uart driver)

muos_error_rx_parity
parity error detected

muos_error_txqueue_overflow
To much data to send (TXQUEUE)

muos_error_eeprom_busy
eeprom busy

muos_error_eeprom_verify
eeprom verification failed

muos_error_configstore_locked
configstore operation in process

muos_error_configstore_invalid
configstore has a problem (check status)

muos_error_stepper_noinit
not initialized

muos_error_stepper_state
action not possible from current state

muos_error_stepper_range
some parameter out of range

muos_error_stepper_noslot
no position match slot

muos_error_stepper_slope
no position match slot

muos_error_cppm_frame
received broken cppm frame

muos_error_cppm_hpq_callback
hpq overflow when pushing cppm handler

3.4.2 API

Error Check Macro

```
MUOS_OK(fn)
```

Wraps fn which must be a function call returning a *muos_error* an a check for success or returning the error.

The type used for error codes

```
typedef enum {...} muos_error
```

Query number of pending errors

```
uint8_t muos_error_pending (void)
```

Returns the number of errors which are flagged.

Flagging asynchronous errors

```
void muos_error_set (muos_error err)
void muos_error_set_isr (muos_error err)
```

err
: Errorcode to flag

When *err* is *muos_success* this function just returns. Thus idioms like

```
muos_error_set (function_which_may_return_an_error ());
```

are possible.

When *err* is already flagged, nothing happens.

The *_*isr* function is for contexts where interrupts are disabled.

Returns *err*.

Query the status of a error flag

```
bool muos_error_peek (muos_error err)
```

err
: error code to query

Returns *true* when the error is flagged, *false* otherwise.

Check for errors

```
bool muos_error_check (muos_error err)
bool muos_error_check_isr (muos_error err)
```

err
: error code to query

When *err* was flagged then return true **and** clear that flag, otherwise return *false*.

The *_*isr* function is for contexts where interrupts are disabled.

3.4.3 configuration

MUOS_ERRORFN
: Name of the user-defined function which is called when am error is pending.

Chapter 4

Drivers

4.1 State Machines

MµOS provides a infrastructure for implementing generic state machines. One can For this the user has to define a list of all possible states and then define functions for the transistions between these states.

While the list of all possible states is global, one can define multiple independent state machines. The transition implementation will ensure that only valid states appear in each machine.

Each state machine stores a few parameters to queue continutation states. This allows the implementation of nested states which return to a calling state or sequencing states in some predefined order.

When more than one state machine is defined an user defined *extra* data member is passed around for identifying the actual state machine.

When multiple state machines are defined they are identified by index.

4.1.1 Example

Prepare the file *states.def*, configure the *Makefile* with -DMUOS_SM_DEF=*states.def* and -DMUOS_SM_NU

states.def

```
#define MUOS_SM_STATES          \
        STATE(INIT)             \
        STATE(ONE)              \
        STATE(TWO)
```

provide enter/leave functions (empty bodies for example):

statemachine.c

```
muos_error
state_INIT_leave (enum muos_sm_state trans[4])
{
  return muos_success;
}

void
state_INIT_enter (void)
{
}
```

```
muos_error
state_ONE_leave (enum muos_sm_state trans[4])
{
  return muos_success;
}

void
state_ONE_enter (void)
{
}

muos_error
state_TWO_leave (enum muos_sm_state trans[4])
{
  return muos_success;
}

void
state_TWO_enter (void)
{
}
```

Then initialize the state machine within the MUOS_INIT function or somewhere else within the application and eventually call state transitions. Error returns left out in this example:

```
void
init (void)
{
  muos_interrupt_enable ();
  MUOS_SM_INIT (INIT);

  ...

  MUOS_SM_CHANGE(ONE);
}
```

4.1.2 API

States Enumeration

```
enum muos_sm_state
   STATE_NONE,
   MUOS_SM_STATES...
 }
```

A global list of all defined states. The user has to supply a file which defines MUOS_SM_STATES. This gets x-macro expanded for various things. Note that all states get prefixed with *STATE_* not the usual *MUOS_** prefix (these are user application defined states anyway).

STATE_NONE gets always defined as the first member implicitly. This is used to flag the state machine when it is not yet initialized or a state transition is in progress.

States Transition Functions

```
typedef muos_error (*const state_leave_fn)(enum muos_sm_state params[4]);
typedef void (*state_enter_fn)(void);

typedef muos_error (*const state_leave_fn)(enum muos_sm_state params[4], intptr_t extra);
typedef void (*state_enter_fn)(intptr_t extra);
```

For each state the user has to define an enter and a leave function. The MµOS state machine driver calls those for changing states.

When MUOS_SM_NUM is greater than one, then state transition functions take an *extra* parameter which passes user defined data and wont be changed by the state machine driver.

This functions need to be named *state_STATE_enter* and *state_STATE_leave*, where *STATE* is the state as defined above (without the *STATE_* prefix).

The *leave* function will be called first with the intended state transition in the *params* field. *params[0]* is the state to be changed to, *params[1..3]* are user defined subsequent state parameters. The *leave* function shall validate the possible state transition and may return an error when the transition is denied. Otherwise it should clean the current state up and return *muos_successs*. Note that the *current* state might not be fully initialized when the *enter* function did not complete and changes into a new state immediately. When the *leave* function returns any error the state stays unchanged.

The *enter* function is called to finish the state transition and should set up the new state. Errors on enter shall be handled by changing into another error-handling state.

State Machine Initialization

```
#define MUOS_SM_INIT(sm, extra, initialstate, ...)

muos_error
muos_sm_init (uint8_t sm, enum muos_sm_state params[4], intptr_t extra);

#define MUOS_SM_INIT(initialstate, ...)

muos_error
muos_sm_init (uint8_t sm, enum muos_sm_state params[4]);
```

An uninitialized state machine is initially at *STATE_NONE* and must be initialized to a first state. Again here the API is different depending on the number of state machines to be defined with MUOS_SM_NUM.

When there is more than one state machine one must identify the state machine with the *sm* index and an initial *extra* member gets passed and initialized. Later this *extra* member gets passed unaltered to to the state transition functions.

MUOS_SM_INIT() shall be used to initialize a state machine. Adds some convenience over calling the *muos_sm_init()* function directly. *MUOS_SM_INIT()* takes a number of optional arguments for the parameter array which will be initialized to *STATE_NONE* if not given. The *STATE_* prefix must be omitted here as it gets automatically applied.

The initialization calls the initial states enter function synchronously, on successful return the state machine is fully initialized and running.

Initialization returns an error when *sm* or *initialstate* is out of range. The state stays at *STATE_NONE* then.

State Transition

```
#define MUOS_SM_CHANGE(sm, newstate, ...)

#define MUOS_SM_CHANGE(newstate, ...)

muos_error
muos_sm_change (uint8_t sm, enum muos_sm_state params[4]);
```

A state transition on a initialized state machine will be initiated by calling the *MUOS_SM_CHANGE()* macro. Again depending on the MUOS_SM_NUM the API may require the index of the state machine to change. The first *newstate* parameter is the state one would like to change to. Any optional subsequent parameters will passed in the params array or initialized to *STATE_NONE* if not given.

State changing will validate it's arguments then call the *leave* function of the current state. When this succeeds the state is temporally set to *STATE_NONE* to flag that a state transition is in progress and the state enter function for the gets appended to the hpq.

On scheduling before the *enter* function for the new state gets called the current state and params will be set to the new state.

Tool Functions

```
enum muos_sm_state*
muos_sm_params_get (uint8_t sm);

enum muos_sm_state
muos_sm_get (uint8_t sm);

#ifdef MUOS_SM_NAMES
const char __flash *
muos_sm_name_get (uint8_t sm);
#endif

bool
muos_sm_ready (intptr_t sm);
```

muos_sm_params_get() queries the params array for the state machine.

muos_sm_get() returns the current state.

muos_sm_name_get() returns a textual representation of the current state.

muos_sm_ready() returns true when the state is not *STATE_NONE*. For use as predicate to *muos_wait()*

4.1.3 configuration

MUOS_SM
If defined, the state machine driver gets compiled in

MUOS_SM_DEF
The file which defines the states.

MUOS_SM_NAMES
Enable storing the textual representation of state names within the program.

MUOS_SM_NUM
Number of state machines. :

4.2 Serial

Communication over serial interfaces is split into serveral parts. On the lowest level is a UART driver which operates directly on the hardware.

Above that is a library which provides output routines for many types. MµOS does not implement the typical stdio like facilities or printf like format strings. This ensures the required flexibility and extensibility which wouldn't be possible in a standard approach.

One of the most distinct features is the TXQUEUE. This is an additional queue which uses a tagged binary representation for the data to be transmitted over serial.

4.2.1 Hardware UART

The UART runs asynchronously driven by interrupts. Data is transferred byte by byte with immediate cyclic buffers. Where supported the transmitter and receiver can be disabled independently.

Access to the UART is always non-blocking, requests which can't be served returning or flagging an error.

At startup the receiver is in desynched state and waits for **\r** to appear in the stream where it changes into the synchronized state and data can be read. This ensures that only complete lines are returned to the program. Details can be be configured. In future it will also synchronize when the line is idle.

I/O comes in blocking and non-blocking variants each can be called independently for TX and RX. Blocking I/O means that the job making the I/O is put on hold and the scheduler is called recursively with *muos_wait()*. There can be only one non-blocking (per direction) I/O pending at any time. Further attempts to do I/O will result in a *muos_tx_blocked* or *muos_rx_blocked* error.

Blocking I/O is in some cases more convenient but needs more space on the stack and few more errors can happen. It still works very well with small buffers which may outweigh the stack costs. Over/under-runs of the buffers can never happen.

Non-Blocking I/O is easier to use when it is used exclusively, then only errors for buffer over/under-runs need to be handled, but the program logic can become more complicated when a lot data needs to be transferred as some requests might be larger than the buffers.

There is also support for registering a function which gets pushed onto the *hpq* when characters ready for reading. MµOS comes with a line editor which can be used in this place.

4.2.1.1 API

Sending data

```
muos_error muos_serial_tx_nonblocking_byte (uint8_t b);
muos_error muos_serial_tx_blocking_byte (uint8_t b);
muos_error muos_serial_tx_byte (uint8_t data)
```

`data`
The byte to send

Pushes a single byte on the TX buffer.

muos_serial_tx_nonblocking_byte
Will not block. In case of error one of the following errors gets returned:

muos_error_tx_buffer_overflow
Transmission buffer is full

muos_error_tx_blocked
There is already a blocking write pending

muos_serial_tx_blocking_byte
Waits until data can be send, entering the scheduler recursively. May return one of the following errors:

muos_warn_sched_depth
Scheduler depth exceeded.

muos_error_tx_blocked
There is already a blocking write pending

muos_warn_wait_timeout
TX got stuck. The timeout is calculated internally depending on baudrate so that some chars should been send. When this error happens something got seriously wrong and the TX doesn't send any data.

muos_serial_tx_byte
Picks one of the functions above, depending on configuration.

All calls return *muos_success* on success or errors as noted above.

Reading data

```
uint16_t muos_serial_rx_nonblocking_byte (void)
uint16_t muos_serial_rx_blocking_byte (muos_shortclock timeout)
uint16_t muos_serial_rx_byte (void)
```

timeout
Time to wait for blocking reads

Pops and a byte from the receive buffer. Zero or positive return value is a successful read from the buffer. Negative return indicates an error by the negated error number. Note that the UART driver may flag asynchronous errors too.

muos_serial_rx_nonblocking_byte
Will not block. Following errors can happen:

muos_error_rx_buffer_underflow
No data available for reading

muos_error_rx_blocked
There is already a blocking read pending

muos_serial_rx_blocking_byte
Waits until data becomes available, entering the scheduler recursively. A timeout of 0 means infinite waits.

Following errors can happen:

muos_warn_sched_depth
Scheduler depth exceeded.

muos_error_rx_blocked
There is already a blocking read pending

muos_warn_wait_timeout
No data received within *timeout*.

muos_serial_rx_byte
Picks one of the functions above, depending on configuration. The timeout for the blocking case defaults to infinite waits.

Return a character value or a negated error as noted above.

RX Callback type

```
typedef bool (*muos_serial_rxcallback)(void)
```

The type for the user-defined *MUOS_SERIAL_RXCALLBACK* function. This function gets called from the *hpq* when there is data data on the RX buffer. When it does not consume all data will called again when it returns *true*. When it returns *false* it will only be called again when **new** data is available on the buffer.

4.2.1.2 configuration

MUOS_SERIAL
Enable Serial

MUOS_SERIAL_BAUD
Baudrate for the serial interface

MUOS_SERIAL_HW
Serial hardware to use

MUOS_SERIAL_RXBUFFER
length of the receive buffer

- 0 disables receiving data
- >1 creates a buffer of that size

MUOS_SERIAL_RXCALLBACK
Function to be called from *hpq* to handle the RX buffer data

MUOS_SERIAL_RXSYNC
The receiver stays in desynced state unti the given character got received

MUOS_SERIAL_RX_BLOCKING
Make the RX blocking by default.

MUOS_SERIAL_TXBUFFER
length of the transmission buffer

- 0 disables sending data
- >1 creates a buffer of that size

MUOS_SERIAL_TX_BLOCKING
Make the TX blocking by default.

4.2.2 I/O Library

MµOS comes with a library to do formatted text output over serial. This library does not use printf alike format strings but provides functions for printing each data type. Actually there are 2 implementations with the same API but different functionality. The first one pushes data directly to the transmission buffer while the second one has its own data queue.

Note
While the basic variant and the TXQUEUE have the same API, not all functions are implemented for both variants. This will be fixed in future as far it makes sense.

Basic variant Pushing data directly out makes the code pretty small and the linker will only link functions which are actually used. The downside is that the TX buffer needs to hold every byte to be printed. When it overruns an error is reported, but the output might be truncated. This is suitable for small targets with little I/O demands, as long one can provide a big enough TX buffer.

TXQUEUE Uses a *tagged* queue as front-end for the TX buffer. Tagged means that the data can be stored in binary form with some tag about the type in front. The I/O library then also manages a job on the hpq which does the transfer from the txqueue to he txbuffer. When the TXQUEUE is in use the TX buffer can be configured to hold only few bytes.

The implementation ensures that the most compact form will be stored. This can dramatically reduce the memory footprint for the queue. For example numbers are represented in the smallest binary form possible, constant strings can be stored in Flash ROM and are only referenced from the TXQUEUE.

This advantage comes with some weight on the flash side, the TXQUEUE implementation currently weight over 2k of code, still whenever enough flash space is available it should be the considered.

Another advantage is that operations on the TXQUEUE are atomic. Data gets never truncated when the queue is full and an error is returned, either the whole thing gets pushed or nothing in case of an error. Future plans include to add some begin/commit transactions support too.

4.2.2.1 API

muos_output_base (uint8_t)
set numeric base for the next integer conversion (2-36)

muos_output_char (char)
a single character

muos_output_csi_char (const char)
CSI followed by a single character

muos_output_csi_cstr (const char*)
CSI followed by a c-string

muos_output_csi_cstr_P (”literal”)
puts ”literal” into flash section and uses `muos_output_csi_fstr` for printing it

muos_output_csi_fstr (muos_flash_cstr)
CSI followed by a c-string stored in flash ROM

muos_output_cstr (const char*)
a c-string

muos_output_cstr_P (”literal”)
puts ”literal” into flash section and uses `muos_output_fstr` for printing it

muos_output_cstr_R (const char*)
c-string by reference

muos_output_cstrn (const char*, uint8_t)
c-string with given maximal length

muos_output_cstrn_R (const char*, uint8_t)
c-string by reference with given maximal length

muos_output_fstr (muos_flash_cstr)
c-string stored in flash ROM

muos_output_int16 (int16_t)
signed numeric value

muos_output_int32 (int32_t)
signed numeric value

muos_output_int64 (int64_t)
signed numeric value

muos_output_int8 (int8_t)
signed numeric value

muos_output_intptr (intptr_t)
signed numeric value of a pointer

muos_output_mem (const uint8_t*, uint8_t)
raw memory with given address and length

muos_output_nl (void)
system dependent newline sequence

muos_output_pbase (uint8_t)
set default numeric base (2-36)

muos_output_pupcase (bool)
set default digit representation

muos_output_uint16 (uint16_t)
unsigned numeric value

muos_output_uint32 (uint32_t)
unsigned numeric value

muos_output_uint64 (uint64_t)
unsigned numeric value

muos_output_uint8 (uint8_t)
unsigned numeric value

muos_output_uintptr (uintptr_t)
unsigned numeric value of a pointer

muos_output_upcase (bool)
set digit representation for the next integer conversion

4.2.2.2 configuration

MUOS_SERIAL_TXQUEUE
length of the txqueue

- 0 disables the txqueue
- >1 creates a txqueue of the given size

4.2.3 Lineedit

For building command line interfaces, mµOS comes with a highly configurable line editor. Over time this is planned to become a complete framework for controlling the program.

One can enable it by choosing it as callback function of the UART receiver `MUOS_SERIAL_RXCALLBACK=muos_lineedit`. Then the line editor presents a user configurable callback which gets called when a line got terminated.

The Line editor can be configure to handle UTF8 encoded text. This only adds moderate size to the code but causes more serial traffic since editing often forces a complete line redraw. When UTF8 support is disabled and only 7-bit ASCII encoding is supported.

Most common line commands are supported. The is cursor moved with cursor keys or vi keys, other editing keys (Home, End, Backspace, Delete, Overwrite) are supported.

There is a very simple line recall handler which lets one restore the previous line when no edits happened yet (Cursor up) or clear the current line (cursor down). This recall feature is very lightweight and needs only one additional byte of RAM and little over 100 bytes program space.

4.2.3.1 configuration

MUOS_LINEEDIT_BUFFER
capacity of the linedit buffer

MUOS_LINEEDIT_CALLBACK
User defined function to call when a line from lineedit is done (return pressed)

MUOS_LINEEDIT_RECALL
Enable lineedit recall

MUOS_LINEEDIT_UTF8
Enable UTF8 support for lineedit

4.3 EEPROM

There are two drivers available for storing data within the microcontrollers EEPROM. The lowlevel *EEPROM* driver allows asynchronous basic access to the eeprom data. Including reading, writing, verifying and erasing data.

The higher level *configstore* driver implements a more robust, fault tolerant and wear leveling algorithm on top of the *EEPROM* facilities.

4.3.1 Low level EEPROM access

Implements basic EEPROM operations with reasonable fine grained control of actual semantics. Writes are asynchronous, reading can be configured to do batched access. Upon completion of a request a user supplied callback can be called. This callback should also check if any error happend.

Asynchronous errors can be:

error_hpq_overflow, error_bgq_overflow
Could not schedule the completion callback.

error_eeprom_verify
A verification failed (*write*, *writeverify*, *is_erased*).

The standard write method does a *smart* write, as it only erases and writes when necessary. There is also a configureable option for retrying a few times writes when write verification failed before returning an error.

Since EEPROM wears out over time overly frequent writing should be avoided. One safety measure against programming bugs writing data in a tight loop is to add a hard delay before the write. This can be configured but should only be enabled in debug builds.

By its nature EEPROM is considered slow and less timing critical, thus it prefers to schedule callbacks on the BGQ when available.

4.3.1.1 API

The parameters used for accessing the eeprom are orthogonal though all access functions

`address`
start address in ram

eeprom
start address in EEPROM

size
size in bytes of for the operation

complete
callback function called upon completion (also on failure). May be NULL.

The access function returns *error_eeprom_busy* when they can not start an asynchronous job. All other errors are set asynchronously and should be handled in the complete callback.

Reading

```
muos_error muos_eeprom_read (void* address,
                             uintptr_t eeprom,
                             size_t size,
                             muos_eeprom_callback complete)
```

Reads the EEPROM.

Writing

```
muos_error muos_eeprom_write (void* address,
                              uintptr_t eeprom,
                              size_t size,
                              muos_eeprom_callback complete)

muos_error muos_eeprom_writeerase (void* address,
                                   uintptr_t eeprom,
                                   size_t size,
                                   muos_eeprom_callback complete)

muos_error muos_eeprom_writeverify (void* address,
                                    uintptr_t eeprom,
                                    size_t size,
                                    muos_eeprom_callback complete)

muos_error muos_eeprom_writeonly (void* address,
                                  uintptr_t eeprom,
                                  size_t size,
                                  muos_eeprom_callback complete)
```

Transfers data from memory to eeprom.

muos_eeprom_write
Smart write, erase/write only when necessary. Fastest when existing content with only few changes is overwritten. Reduces EEPROM wear.

muos_eeprom_writeerase
Erases data before writing. Faster than erasing the block first.

muos_eeprom_writeverify
Erases data before writing. Verifies after write.

muos_eeprom_writeonly
Only writes (clears bits) without erasing.

Erasing

```
muos_error muos_eeprom_erase (uintptr_t eeprom,
                              size_t size,
                              muos_eeprom_callback complete)
```

Erases the given range.

Verifying

```
muos_error muos_eeprom_verify (void* address,
                               uintptr_t eeprom,
                               size_t size,
                               muos_eeprom_callback complete)
```

Compares a block of memory with eeprom contents. Aborts on first error, calling *complete*.

Xoring

```
muos_error muos_eeprom_xor (uint8_t* address,
                            uintptr_t eeprom,
                            size_t size,
                            muos_eeprom_callback complete)
```

XOR'es the given range with address pointing to a single uint8_t.

Cyclic Redundancy Check

```
muos_error muos_eeprom_crc16 (uint16_t* address,
                              uintptr_t eeprom,
                              size_t size,
                              muos_eeprom_callback complete)
```

Caclculates the CRC16 (configured by MUOS_EEPROM_CRC16_FN) of the given range and stores it at address. NOTE: the initial value of **address* must be set by the user.

Only available when MUOS_EEPROM_CRC16_FN is configured.

Check for erased

```
muos_error muos_eeprom_is_erased (uintptr_t eeprom,
                                  size_t size,
                                  muos_eeprom_callback complete)
```

Checks if the given range is erased.

Query State

```
enum muos_eeprom_mode muos_eeprom_state (void)
```

Polls the state of the driver.

Returns 0 (MUOS_EEPROM_IDLE) when no operation is in progress.

4.3.1.2 configuration

MUOS_EEPROM
Enable the low level asynchronous EEPROM driver

MUOS_EEPROM_CRC16_FN
Enables the crc16 api, must name a function to call with `uint16_t crc16_update (uint16_t crc, uint8_t data)` as prototype the avr-libc <*util/crc16.h*> can be used

MUOS_EEPROM_CRC16_INCLUDE
Define an extra include for the CRC16 function definition.

MUOS_EEPROM_CRC16_INIT
the initial value used for the crc16 caclulation.

MUOS_EEPROM_DEBUG_WDELAY
Delay program execution by roughly this much milliseconds before each write. This ensures programming errors won't wear out the eeprom in no time.

Warning
Development option only, uses busy loop delay and will halt processing.

4.3.2 Configstore

The configstore comes with different facilities.

1. A flat representation of configuration values which can be queried from the application.
2. A mapping from strings to the value to be used for CLI and other protocol implementations which are text based.
3. A backend for loading and saving this configuration in a very (configureable) reliable/atomic way. By using some journaled write strategy the wear of the EEPROM will be leveled/mitigated.

With all bells and whistles enabled it is failure tolerant to power glitches and damaged eeprom cells.

4.3.2.1 API

Configuration Description

```
#define CONFIGSTORE_DATA             \
  CONFIGSTORE_ENTRY(type, ary, name) \
  ...
```

type
C type of the data.

ary
Size for arrays. 0 for single variables. Otherwise one of 2, 3, 4, 5, 6, 7,8, 16, 32, 64, 128, or ARRAY(n) to define the size of an array.

name
C identifier for the variable.

Configuration variables are defined by a C-Preprocessor defined DSL in a single included file. The *MUOS_CONFIGSTORE_INCLUDE* configuration from the *Makefile* must point to this file.

The user defines *CONFIGSTORE_DATA* to a sequence of *CONFIGSTORE_ENTRY(type, ary, name)* definitions. MµOS uses this to expand the provided data into various datastructures.

Loading and Saving

```
muos_error
muos_configstore_load (muos_configstore_callback callback)

muos_error
muos_configstore_save (muos_configstore_callback callback)
```

`callback`
function called on completion.

Both functions return *muos_error_configstore_locked* in case of an error. Other errors should be handled in *callback*.

Access

```
const struct muos_configstore_data*
muos_configstore_lock (void)

struct muos_configstore_data*
muos_configstore_wlock (void)

void
muos_configstore_unlock (const struct muos_configstore_data** lock)

void
muos_configstore_unwlock (struct muos_configstore_data** lock)

struct muos_configstore_data*
muos_configstore_initial (void)
```

The configuration data implements a simple locking scheme with multiple readers or a single writer. Each successful lock must be paired with a unlock, no further consistency checks are made!

Locking works only on valid data.

There are approx. 250 read locks available. Exceeding this number makes the lock fail.

muos_configstore_lock ()
Checks for availability of the configstore. On success it places a read lock on the configstore and returns a pointer to a const *muos_configstore_data* data structure which holds all the defined elements. On failure `NULL` is returned and one may inspect the configstore status. No mutations must be made to the data.

muos_configstore_wlock ()
Checks for availability of the configstore. On success it places a write lock on the configstore and returns a pointer to a *muos_configstore_data* data structure which holds all the defined elements. On failure `NULL` is returned and one may inspect the configstore status. The write lock blocks all other access to the configstore and may modify its contents.

muos_configstore_initial ()
Works only when the configstore is *invalid*. Places a write locks on the data which must be unlocked afterwards. This is used when the configstore is uninitialized/prime or damaged to populate it with defaults.

muos_configstore_unlock () muos_configstore_unwlock ()
Frees the lock obtained by *muos_configstore_lock()*, *muos_configstore_wlock()* or *muos_configstore_ini*
(). Care must be taken that lock is matched by a unlock.

4.3.2.2 configuration

MUOS_CONFIGSTORE
Enable the configuration store (EEPROM) driver

MUOS_CONFIGSTORE_INCLUDE
The file which defines the configuration variables

MUOS_CONFIGSTORE_OFFSET
Use EEPROM addresses from OFFSET upward, user is responsible that this is page aligned

MUOS_CONFIGSTORE_SIZE
Use this much bytes at most

4.4 Stepper motors

Stepper driver for motor controllers taking pulse/direction signals. Allows for continious movements of stepper motors with speed profiles.

4.4.1 Speed Marks

min_speed
implicit, the slowest speed the hardware can generate.

cal_speed
speed used for calibration/zeroing.

slow_speed
motors must not loose steps when started/stopped to this speed. minimum speed for absolute positioning with acceleration/deceleration. maximum speed for relative positioning which does not accelerate/decelerate.

max_speed
absolute maximum speed which will not exceeded. motors must run stable up to this speed.

4.4.2 Speed Profiles

The Driver generates a slope which accelerates towards *max_speed* with decreasing acceleration (steppers have less torque at higher speeds), then running a stretch on constant (max.) speed, then going into deceleration until the target speed is reached. Finally some steps are done at the target speed. As soon a slope is loaded and executed the next slope can be generated in background to allow fluent movement.

4.4.3 General Operation Overview

Before one can start moving a steppers it needs some preparations. If configured then one needs to enable (energize) the steppers first. This may already be blocked by some external input (Emergency Switch).

An energized stepper still does not know it’s position, further some configuration for the machine parameters (speed limits, prescaler, acceleration/deceleration etc.) is required for all but the simplest raw movements.

With steppers enabled and configuration given the stepper can do slow relative movements. But the origin is not known yet. For this the stepper needs to be zeroed (by probing limit switches or similar). When zeroing is done the steppers becomme fully armed and all movement types are possible.

When a stepper completes one or multiple movements it will usually stay in the highest possible state, staying enabled and keep its position until they become explicitly disabled.

Raw movemnt are provided for low level calibration taskes. Care must been taken when using *raw* movements because these are not constrained by machine parameters and may cause overload/damages.

```
document that steppers *must* move before zeroing to lock in phase
```

4.4.4 Slope Preparation

Since the generation of slope parameters is relatively expensive it can not be done at the instant the stepper shall start moving but need to be prepared beforehand.

For this the set of slope parameters is doublebuffered and a callback function can be invoked to generate the next set of parameters as soon the current ones get activated.

It is also possible to pregenerate and cache slope parameters for common, esp. short movements.

4.4.5 Moving multiple Axis in sync

4.5 CPPM

Parses sum-signals from a RC Receiver. The results are stored in global array and a function can be pushed onto the *hpq* whenever a frame got received. The CPPM driver uses the *input capture* feature from the main clock. Thus one has to select a timer which supports this feature when using CPPM. Using input capture ensures the most precise timing.

Note
There is a plan to implement an alternative CPPM driver using pin-change interrupts. This will allow some flexibiltiy in choosing the timer for the clock and free the input capture for other tasks.

There are 2 configurable ways how the driver can store the results for the user, raw timer ticks or cooked values in the range from -125 to 125.

4.5.1 Details

One should configure the minimum and maximum signal lengths. If a signal is to short then `muos_error_cppm_frame` will be flagged, while longer signals mark the start of a new frame.

When *MUOS_CPPM_FRAME_CLOCKSYNC* is enabled, mµOS will adjust the µC speed to the cppm frames. This required *MUOS_CPPM_FRAME* to be configured correctly. This is a way to get even better timing from µC which uses a poor internal oscillator.

In case the *hpq* is full when the driver wants to push the *MUOS_CPPM_CALLBACK* function, the error `muos_error_cppm_hpq_callback` gets flagged.

The callback function will not be called when erroneous frames are received. The user is advised to implement some watchdog to handle this case.

4.5.2 API

Raw values

```
uint16_t muos_cppm_channel_raw[MUOS_CPPM_CHANNELS]
```

Stores the time in timer ticks as measuered for each channel directly, after applying some configureable filter. This gives the most precision but also needs more memory. Must be enabled with *MUOS_CPPM_RAW*.

Raw values

```
int8_t muos_cppm_channel_cooked[MUOS_CPPM_CHANNELS]
```

Stores the channel data as values from -125 to 125 mapping to the range from *MUOS_CPPM_COOKED_MIN* to *MUOS_CPPM_COOKED_MAX*. Little overflows from -128 to 127 are tolerated. Cooked values need less memory and are more stable, but have lower precision. Must be enabled with *MUOS_CPPM_COOKED*.

4.5.3 configuration

MUOS_CPPM
Enable the CPPM Parser for Radio Control Signals

MUOS_CPPM_CALLBACK
function to be pushed to the hpq when a frame got received

MUOS_CPPM_CAPTURE
Select Input-Capture driver

MUOS_CPPM_CHANNELS
Number of CPPM channels

MUOS_CPPM_COOKED
Store processed cppm data.

MUOS_CPPM_COOKED_MAX
Defines the maximum signal length for cooked values

MUOS_CPPM_COOKED_MIN
Defines the minimum signal length for cooked values.

MUOS_CPPM_FRAME
Length of a full CPPM frame, required for clocksync

MUOS_CPPM_FRAME_CLOCKSYNC
Use the MUOS_CLOCK_CALIBRATE to synchronize the µC clock with the CPPM frames

MUOS_CPPM_MAX
Maximum CPPM Pulse length, any longer pulse defines the frame start

MUOS_CPPM_MIN
Minimum CPPM Pulse length

MUOS_CPPM_RAW
Store the raw cppm timing data.

MUOS_CPPM_RAW_FILTER
Filter expression for raw channel data

4.6 Debug

Subsystem for using few GPIOs as debug outputs for mµOS internals and user defined channels.

4.6.1 API

Debug GPIO Macros

```
MUOS_DEBUG_Cx_ON
MUOS_DEBUG_Cx_OFF
MUOS_DEBUG_Cx_TOGGLE
```

x
Debug channel (1-4)

Sets the debug channel to the given state.

```
MUOS_DEBUG_INTR_ON
MUOS_DEBUG_INTR_OFF
```

MµOS Interrupt debug channel. User defined interrupts should turn the GPIO on right at the start and off when done.

4.6.2 configuration

MUOS_DEBUG
When defined, the debug driver is enabled

MUOS_DEBUG_BUSY
Uses a GPIO to indicate when the CPU is busy. While sleeping the GPIO is turned off.

MUOS_DEBUG_Cx
MUOS_DEBUG_C1, MUOS_DEBUG_C2, MUOS_DEBUG_C3, MUOS_DEBUG_C4. User defined Debugging channels.

MUOS_DEBUG_ERROR
Uses a GPIO to indicate when an error got set, Turned off when no errors are pending.

MUOS_DEBUG_INTR
Uses a GPIO to indicate when the CPU is handling interrupts.

MUOS_DEBUG_SWITCH
Uses a GPIO to indicate when the scheduler switches to another function, Pin gets toggled!

Chapter 5

Library

Some common data structures and algorithms are implemented in a small run time library which ships with mµOS. This library functions are meant as simple building blocks for other facilities in mµOS. Some things are deliberately left out or simplified, if one wants to use parts of the library he has to be careful about this.

Notably most library calls usually don't disable/enable interrupts and don't check for errors. The user of the library has to implement this.

This is intentionally left out because often one wants more than one single operation atomic. Like pushing two elements on a queue, which should either fail and leave the queue in the old state or succeed with both elements on the queue. This can only be implemented from the callers scope.

5.1 queue

The Queue used for the work queues, storing functions and arguments. The size of the queue must be given at compile time. The implementation works with any size, there is no requirement for the size to be a power of two.

Functions pushed on the queue can take an optional argument along. The current implementation flagging this in MSB of the function pointer. This restricts the current implementation to 64k address space (AVR, word addressed). Future implementation will lift this restriction.

The type used for indexing can be configured to 4, 8 or 16 bits. For extremely small microprocessors using 4 bits only makes the queue management data fit into a single byte. Queue size is constrained to 16 elements then. Passing arguments to functions require one additional element.

The API is made from a small layer of macros above the underlying functions which take care of passing the correct type and size along.

5.1.1 API

Queue definition

```
MUOS_QUEUEDEF(size)
```

size
: number of elements

Macro defining the type of a queue for the given size. To instantiate a queue use `MUOS_QUEUEDEF(16) myqueue;` for example.

Queue Initialization

```
void muos_queue_init (struct muos_queue* queue)
```

queue
pointer to the queue

Initialization is not necessary at startup, it is only required for to reinitialize and delete an existing queue.

Queue operations

```
void muos_queue_pushback (struct muos_queue* queue,
                          const muos_queue_size size,
                          muos_queue_function func)

void muos_queue_pushback_arg (struct muos_queue* queue,
                              const muos_queue_size size,
                              muos_queue_function func,
                              intptr_t arg)

void muos_queue_pushfront (struct muos_queue* queue,
                           const muos_queue_size size,
                           muos_queue_function func)

void muos_queue_pushfront_arg (struct muos_queue* queue,
                               const muos_queue_size size,
                               muos_queue_function func,
                               intptr_t arg)

intptr_t muos_queue_pop (struct muos_queue* queue,
                         const muos_queue_size size)
```

queue
the queue

size
size of the queue

func
function pointer

arg
inptr_t argument

- `muos_queue_pushback ()` pushes *func* onto the back of the queue
- `muos_queue_pushback_arg ()` pushes *func* with *arg* onto the back of the queue
- `muos_queue_pushfront ()` pushes *func* onto front of the queue
- `muos_queue_pushfront_arg ()` pushes *func* with *arg* onto the front of the queue
- `muos_queue_pop ()` removes the first argument from the queue

Queue information

```
muos_queue_size muos_queue_free (struct muos_queue* queue, const muos_queue_size size)
```

queue
the queue

size
size of the queue

Returns the number of free elements in the queue.

5.1.2 configuration

MUOS_QUEUE_INDEX
the bits used for indexing queues comes in 3 variants

- 4 bits allow only small queues for up to 16 elements, use only for really small targets
- 8 bits allow queues for up to 256 elements, default
- 16 bits allow huge queues, use only when really required

5.2 spriq

Priority queue for storing functions scheduled by time (clpq). This implements a sliding window, times (priorities) are stored in future to a given base. Times smaller (due integer overflows) than the base will compare as greater than the base in the queue, appending them at the end and thus define the sliding window.

This sliding window implementation was chosen because in most cases one only wants to schedule jobs within relative short times into the future. Storing a full time-stamp (which can be up to 80bit) within the priority queue would be a huge waste of memory.

5.2.1 API

Types

```
struct muos_spriq_entry
  muos_spriq_priority when;
  muos_spriq_function fn;
}

typedef void (*muos_spriq_function)(const struct muos_spriq_entry* event)
```

The type for functions stored in a spriq. This function is called with the time it was to be scheduled. The scheduler passes the spriq entry to the function, this makes it simple to schedule repeating jobs, where this *event→when* just becomes the new *base* for next push.

Initialize a spriq

```
void muos_spriq_init (struct muos_spriq* spriq)
```

spriq
pointer to the spriq

Initialization is not necessary at startup, it is only required for to reinitialize and delete an existing queue.

Push a function

```
void muos_spriq_push (
       struct muos_spriq* spriq,
       muos_spriq_priority base,
       muos_spriq_priority when,
       muos_spriq_function fn)
```

spriq
Pointer to the spriq

base
Base priority

when
Offset to base for the priority

fn
Function to push

base must be smaller or equal to the smallest (first) element in the queue. For times this is usually *muos_now()* or the time the event was scheduled. *when* can cover the full range of the priority data type.

Pop element

```
void muos_spriq_pop (struct muos_spriq* spriq)
```

spriq
Spriq where to pop from

No return, no error checking!

5.2.2 configuration

MUOS_SPRIQ_INDEX
Type to keep track of the size of the spriq. `uint8_t` for up to 255 entries, `uint16_t` for up to 65k entries.

MUOS_SPRIQ_TYPE
Type used for the *priorities* in spriq.

5.3 cbuffer

Cyclic byte buffer used for I/O queues. Normally used as queue, but has functions to pop functions from the end and peek and poke at arbitrary positions.

5.3.1 API

Cbuffer definition

```
MUOS_CBUFFERDEF(size)
```

size
number of elements

Macro defining the type of a cbuffer for the given size.

Cbuffer Initialization

```
void muos_cbuffer_init (struct muos_cbuffer* cbuffer)
```

cbuffer
pointer to the cbuffer

Initialization is not necessary at startup, it is only required for to reinitialize and delete an existing queue.

Cbuffer API Macros

```
MUOS_CBUFFER_SIZE(cbuffer)
MUOS_CBUFFER_FREE(cbuffer)
MUOS_CBUFFER_USED(cbuffer)
MUOS_CBUFFER_PUSH(cbuffer, value)
MUOS_CBUFFER_POP(cbuffer)
MUOS_CBUFFER_RPOP(cbuffer)
MUOS_CBUFFER_POPN(cbuffer, n)
MUOS_CBUFFER_PEEK(cbuffer, n)
MUOS_CBUFFER_POKE(cbuffer, n, value)
```

cbuffer
the cbuffer as defined with `MUOS_CBUFFERDEF()`

value
byte (`uint8_t`) value

n
number or position of elements

- `MUOS_CBUFFER_SIZE(cbuffer)` returns the size
- `MUOS_CBUFFER_FREE(cbuffer)` returns how many bytes are free
- `MUOS_CBUFFER_USED(cbuffer)` returns how many bytes are used
- `MUOS_CBUFFER_PUSH(cbuffer, value)` pushes a byte to the end
- `MUOS_CBUFFER_POP(cbuffer)` pops and returns the first byte
- `MUOS_CBUFFER_RPOP(cbuffer)` pops the last byte (no return)
- `MUOS_CBUFFER_POPN(cbuffer, n)` pops *n* bytes frome the begin (no return)
- `MUOS_CBUFFER_PEEK(cbuffer, n)` returns the byte at position *n*
- `MUOS_CBUFFER_POKE(cbuffer, n, value)` changes the byte at position *n* to *value*

5.3.2 configuration

MUOS_CBUFFER_INDEX
type used to the cyclic buffer length and indexing

5.4 utf8

Few routines for handling utf8 encoded strings.

5.4.1 API

Character Tests

```
bool muos_utf8ascii (const char c)
bool muos_utf8start (const char c)
bool muos_utf8cont (const char c)
```

c
character to check

Returns true

- `muos_utf8ascii (c)` when *c* is a latin1 character
- `muos_utf8start (c)` when *c* is the begin of a multibyte sequence
- `muos_utf8cont (c)` when *c* is the a continuation of a multibyte sequence

String length

```
size_t muos_utf8len (const char* str)
```

str
zero terminated c-string

str must not be a continuation byte

Returns the length of a utf-8 encoded string in characters.

Character size

```
uint8_t muos_utf8size (const char* char)
```

char
character to analyze

char can point into the middle or to the end of a multibyte sequence, but the sequence must have a proper start byte.

Returns the size in bytes of the given multibyte character sequence.

Appendix A

LICENSE

MµOS is licensed under GPLv3. For questions about licensing details and possible relicensing conditions contact the author Christian Thäter <ct@pipapo.org>.


```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
```

know their rights.

  Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

  Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

  0. Definitions.

  ”This License” refers to version 3 of the GNU General Public License.

  ”Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

  ”The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as ”you”. ”Licensees” and ”recipients” may be individuals or organizations.

  To ”modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a ”modified version” of the earlier work or a work ”based on” the earlier work.

  A ”covered work” means either the unmodified Program or a work based on the Program.

  To ”propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a

computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

  To ”convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays ”Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The ”source code” for a work means the preferred form of the work for making modifications to it. ”Object code” means any non-source form of a work.

  A ”Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

  The ”System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A ”Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

  The ”Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

  The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

  The Corresponding Source for a work in source code form is that

same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

  When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to ”keep intact all notices”.

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an ”aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this

conveying of source, or (2) access to copy the
Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A ”User Product” is either (1) a ”consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, ”normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

”Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the

Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

”Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some

```
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered ”further
restrictions” within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.
```

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An ”entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A ”contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's ”contributor version”.

A contributor's ”essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, ”control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a ”patent license” is any express agreement or commitment, however denominated, not to enforce a patent

(such as an express permission to practice a patent or covenant not to sue for patent infringement). To ”grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. ”Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is ”discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License ”or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD

PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the ”copyright” line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>
This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an ”about box”.

You should also get your employer (if you work as a programmer) or school,

if any, to sign a ”copyright disclaimer” for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs. If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library. If this is what you want to do, use the GNU Lesser General
Public License instead of this License. But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.