Semaphore APIs

```
int     sem_destroy(sem_t *  semaphore_handle);
int     sem_init(sem_t *  semaphore_handle, int pshared, unsigned int value);
int     sem_post(sem_t *  semaphore_handle);
int     sem_trywait(sem_t *  semaphore_handle);
int     sem_wait(sem_t *  semaphore_handle);
int     px5_sem_extend_init(sem_t *  semaphore_handle, int pshared, unsigned int value,
            semattr_t *  semaphore_attributes);
int     px5_sem_timedwait(sem_t *  semaphore_handle, tick_t  timemout_ticks);
int     px5_semattr_destroy(semattr_t *semaphore_attributes);
int     px5_semattr_getcontroladdr(semattr_t *semaphore_attributes, void **  semaphore_control_address);
int     px5_semattr_getcontrolsize(semattr_t *semaphore_attributes, size_t *  semaphore_control_size);
int     px5_semattr_getname(semattr_t *semaphore_attributes, char **  semaphore_name);
int     px5_semattr_init(semattr_t *semaphore_attributes);
int     px5_semattr_setcontroladdr(semattr_t *semaphore_attributes, void *  semaphore_control_address,
            size_t  semaphore_control_size);
int     px5_semattr_setname(semattr_t *semaphore_attributes, char *  semaphore_name);
```

Memory Pool APIs

```
int     px5_pthread_memory_manager_enable(void);
int     px5_pthread_memory_manager_get(void * (** memory_allocate_pointer)(u_int type, u_long size),
            void (** memory_release_pointer)(u_int type, void *memory_to_release));
int     px5_pthread_memory_manager_set(void * (* memory_allocate_pointer)(u_int type, u_long size),
            void (* memory_release_pointer)(u_int type, void *memory_to_release));
int     px5_pthread_memorypool_allocate(pthread_memorypool_t *  memorypool_handle,
            void **  allocated_memory, size_t  request_size);
int     px5_pthread_memorypool_create(pthread_memorypool_t *  memorypool_handle,
            pthread_memorypoolattr_t *  memorypool_attributes, void *  pool_start, size_t  pool_size);
int     px5_pthread_memorypool_destroy(pthread_memorypool_t *  memorypool_handle);
int     px5_pthread_memorypool_free(void *  allocated_memory);
int     px5_pthread_memorypool_timedallocate(pthread_memorypool_t *  memorypool_handle,
            void **  allocated_memory, size_t  request_size, tick_t  timeout_ticks);
int     px5_pthread_memorypool_tryallocate(pthread_memorypool_t *  memorypool_handle,
            void **  allocated_memory, size_t  request_size);
int     px5_pthread_memorypoolattr_destroy(pthread_memorypoolattr_t *  memorypool_attributes);
int     px5_pthread_memorypoolattr_getcontroladdr(pthread_memorypoolattr_t *  memorypool_attributes,
            void ** memorypool_control_address);
int     px5_pthread_memorypoolattr_getcontrolsize(pthread_memorypoolattr_t *  memorypool_attributes,
            size_t * memorypool_control_size);
int     px5_pthread_memorypoolattr_getname(pthread_memorypoolattr_t *  memorypool_attributes, char ** name);
int     px5_pthread_memorypoolattr_init(pthread_memorypoolattr_t *  memorypool_attributes);
int     px5_pthread_memorypoolattr_setcontroladdr(pthread_memorypoolattr_t *  memorypool_attributes,
            void *  memorypool_control_address, size_t  memorypool_control_size);
int     px5_pthread_memorypoolattr_setname(pthread_memorypoolattr_t *  memorypool_attributes, char *  name);
```

Partition Pool APIs

```
int     px5_pthread_partitionpool_allocate(pthread_partitionpool_t * partitionpool_handle,
            void **  allocated_memory, size_t  request_size);
int     px5_pthread_partitionpool_create(pthread_partitionpool_t *  partitionpool_handle,
            pthread_partitionpoolattr_t *  partitionpool_attributes, void *  pool_start, size_t  pool_size,
            size_t partition_size);
int     px5_pthread_partitionpool_destroy(pthread_partitionpool_t *  partitionpool_handle);
int     px5_pthread_partitionpool_free(void *  allocated_memory);
int     px5_pthread_partitionpool_timedallocate(pthread_partitionpool_t * partitionpool_handle,
            void **  allocated_memory, size_t  request_size, tick_t  timeout_ticks);
int     px5_pthread_partitionpool_tryallocate(pthread_partitionpool_t *  partitionpool_handle,
            void **  allocated_memory, size_t  request_size);
int     px5_pthread_partitionpoolattr_destroy(pthread_partitionpoolattr_t *  partitionpool_attributes);
int     px5_pthread_partitionpoolattr_getcontroladdr(pthread_partitionpoolattr_t *  partitionpool_attributes,
            void ** partitionpool_control_address);
int     px5_pthread_partitionpoolattr_getcontrolsize(pthread_partitionpoolattr_t *  partitionpool_attributes,
            size_t * partitionpool_control_size);
int     px5_pthread_partitionpoolattr_getname(pthread_partitionpoolattr_t *  partitionpool_attributes, char ** name);
int     px5_pthread_partitionpoolattr_init(pthread_partitionpoolattr_t *  partitionpool_attributes);
int     px5_pthread_partitionpoolattr_setcontroladdr(pthread_partitionpoolattr_t *  partitionpool_attributes,
            void *  partitionpool_control_address, size_t  partitionpool_control_size);
int     px5_pthread_partitionpoolattr_setname(pthread_partitionpoolattr_t *  partitionpool_attributes, char *  name);
```

All APIs with leading "px5_" are PX5 RTOS pthread+ extensions.

Constants

CLOCK_REALTIME	O_CREAT
CLOCK_MONOTONIC	O_EXCL
PTHREAD_CANCEL_ASYNCHRONOUS	O_NONBLOCK
PTHREAD_CANCEL_ENABLE	PTHREAD_ALL_EVENTS
PTHREAD_CANCEL_DEFERRED	PTHREAD_ANY_EVENT
PTHREAD_CANCEL_DISABLE	PTHREAD_MUTEX_MAX_NESTING
PTHREAD_CREATE_DETACHED	PTHREAD_PRIO_INHERIT
PTHREAD_CREATE_JOINABLE	PTHREAD_PRIO_NONE
PTHREAD_MUTEX_ERRORCHECK	PTHREAD_PRIO_PROTECT
PTHREAD_MUTEX_NORMAL	SEM_VALUE_MAX
PTHREAD_MUTEX_RECURSIVE	SIG_BLOCK
PTHREAD_PROCESS_SHARED	SIG_UNBLOCK
PTHREAD_PROCESS_PRIVATE	SIG_SETMASK
O_RDWR	

Error Codes

0 (Success for most APIs)
-1 (Error on non-pthread APIs, use errno for detailed error)

EADDRINUSE	EMSGSIZE
EAGAIN	EMVSERR
EAFNOSUPPORT	ENFILE
EALREADY	ENOBUFS
EBADF	ENODEV
EBUSY	ENOMEM
ECONNABORTED	ENOPROTOOPT
ECONNRESET	ENOSPC
EDEADLK	ENOSYS
EFAULT	ENOTCONN
EHOSTUNREACH	EOPNOTSUPP
EINPROGRESS	ESRCH
EINTR	EPERM
EINVAL	ETIMEDOUT
EIO	EWOULDBLOCK
EISCONN	

[RTOS]

PX5 RTOS pthreads+ Programmer's Reference Card

11440 West Bernardo Court, Suite 300, San Diego, CA 92127 | (858) 753-1715 | info@px5rtos.com

PX5 RTOS pthreads+ Programmer's Reference Card

px5rtos.com

Threading APIs

```
Int        pthread_attr_destroy(pthread_attr_t *  thread_attributes);
int        pthread_attr_getdetachstate(pthread_attr_t *  thread_attributes, int *  detach_state);
int        pthread_attr_getstackaddr(pthread_attr_t *  thread_attributes, void **  stack_address);
int        pthread_attr_getstacksize(pthread_attr_t *  thread_attributes, size_t *  stack_size);
int        pthread_attr_init(pthread_attr_t *  thread_attributes);
int        pthread_attr_setdetachstate(pthread_attr_t *  thread_attributes, int detach_state);
int        pthread_attr_setstackaddr(pthread_attr_t *  thread_attributes, void *  stack_address);
int        pthread_attr_setstacksize(pthread_attr_t *  thread_attributes, size_t  stack_size);
int        pthread_cancel(pthread_t thread_handle);
void       pthread_cleanup_pop(int execute);
void       pthread_cleanup_push(void (*cleanup_handler)(void *), void *  argument);
int        pthread_create(pthread_t *  thread_handle, pthread_attr_t *  attr,  void *(*start_routine)(void *), void *arg);
int        pthread_detach(pthread_t thread_handle);
int        pthread_equal(pthread_t first_thread, pthread_t second_thread);
void       pthread_exit(void *  exit_value);
int        pthread_join(pthread_t  thread_handle, void **  value_destination);
pthread_t  pthread_self(void);
int        pthread_setcancelstate(int  new_state, int *  old_state);
int        pthread_setcanceltype(int  new_type, int *  old_type);
void       pthread_testcancel(void);
int        sched_yield(void);
int        px5_pthread_priority_change(pthread_t thread_handle, int new_priority, int *  old_priority);
int        px5_pthread_resume(pthread_t thread_handle);
int        px5_pthread_start(u_long run_time_id, void *  memory_start, u_long memory_size);
int        px5_pthread_stack_check(pthread_t thread_handle, u_long *  minimum_available_stack);
int        px5_pthread_suspend(pthread_t thread_handle);
int        px5_pthread_attr_getcontroladdr(pthread_attr_t *  thread_attributes, void **  thread_control_address);
int        px5_pthread_attr_getcontrolsize(pthread_attr_t *  thread_attributes, size_t *  thread_control_size);
int        px5_pthread_attr_getname(pthread_attr_t *  thread_attributes, char **  name);
int        px5_pthread_attr_getpriority(pthread_attr_t *  thread_attributes, int *  priority);
int        px5_pthread_attr_gettimeslice(pthread_attr_t *  thread_attributes, u_long *  thread_time_slice);
int        px5_pthread_attr_setcontroladdr(pthread_attr_t *  thread_attributes, void *  thread_control_address,
                size_t thread_control_size);
int        px5_pthread_attr_setname(pthread_attr_t *  thread_attributes, char *  name);
int        px5_pthread_attr_setpriority(pthread_attr_t *  thread_attributes, int  priority);
int        px5_pthread_attr_settimeslice(pthread_attr_t *  thread_attributes, u_long  thread_time_slice);
int        px5_pthread_information_get(pthread_t  thread_handle, char **  name,  int * state, int * priority,
                void **  stack_limit, void **  stack_pointer, u_long *  minimum_stack, pthread_t *  next_thread);
```

Time APIs

```
int      clock_getres(clockid_t  clock_id,  struct px5_timespec  *resolution);
int      clock_gettime(clockid_t  clock_id,  struct px5_timespec  *current_time);
int      clock_settime(clockid_t  clock_id,  struct px5_timespec  *new_time);
int      nanosleep(const struct px5_timespec *request_time, struct px5_timespec *remaining_time);
u_int    sleep(unsigned int  seconds);
time_t   time(px5_time_t *return_seconds);
int      usleep(useconds_t  microseconds);
int      px5_pthread_tick_sleep(tick_t ticks_to_sleep);
tick_t   px5_pthread_ticks_get(void);
int      px5_pthread_ticktimer_create(pthread_ticktimer_t *ticktimer_handle, pthread_ticktimerattr_t * attributes,
              void (*expiration_routine)(pthread_ticktimer_t *, void *), void *argument, tick_t  initial_ticks, tick_t
              reload_ticks);
int      px5_pthread_ticktimer_destroy(pthread_ticktimer_t *ticktimer_handle);
int      px5_pthread_ticktimer_start(pthread_ticktimer_t *ticktimer_handle);
int      px5_pthread_ticktimer_stop(pthread_ticktimer_t *ticktimer_handle);
int      px5_pthread_ticktimer_update(pthread_ticktimer_t *ticktimer_handle, tick_t  initial_ticks, tick_t
              reload_ticks);
int      px5_pthread_ticktimerattr_destroy(pthread_ticktimerattr_t *ticktimer_attributes);
int      px5_pthread_ticktimerattr_getcontroladdr(pthread_ticktimerattr_t *  ticktimer_attributes,
              void **  ticktimer_control_address);
int      px5_pthread_ticktimerattr_getcontrolsize(pthread_ticktimerattr_t *  ticktimer_attributes,
              size_t *  ticktimer_control_size);
int      px5_pthread_ticktimerattr_getname(pthread_ticktimerattr_t *  ticktimer_attributes, char ** name);
int      px5_pthread_ticktimerattr_init(pthread_ticktimerattr_t *ticktimer_attributes);
int      px5_pthread_ticktimerattr_setcontroladdr(pthread_ticktimerattr_t *  ticktimer_attributes,
              void *  ticktimer_control_address, size_t  ticktimer_control_size);
int      px5_pthread_ticktimerattr_setname(pthread_ticktimerattr_t *  ticktimer_attributes, char *  name);
```

Condition Variable APIs

```
int      pthread_cond_broadcast(pthread_cond_t *  condition_var_handle);
int      pthread_cond_destroy(pthread_cond_t *  condition_var_handle);
int      pthread_cond_init(pthread_cond_t *  condition_var_handle, pthread_condattr_t *  condition_var_attributes);
int      pthread_cond_signal(pthread_cond_t *  condition_var_handle);
int      pthread_cond_timedwait(pthread_cond_t *  condition_var_handle, pthread_mutex_t * mutex_handle,
              const struct px5_timespec *absolute_time);
int      pthread_cond_wait(pthread_cond_t *  condition_var_handle, pthread_mutex_t *  mutex_handle);
int      pthread_condattr_destroy(pthread_condattr_t *  condition_var_attributes);
int      pthread_condattr_getpshared(pthread_condattr_t *  condition_var_attributes, int *
              process_sharing_designation);
int      pthread_condattr_init(pthread_condattr_t *  condition_var_attributes);
int      pthread_condattr_setpshared(pthread_condattr_t *  condition_var_attributes, int
              process_sharing_designation);
int      px5_pthread_condattr_getcontroladdr(pthread_condattr_t *  condition_var_attributes,
              void ** condition_var_control_address);
int      px5_pthread_condattr_getcontrolsize(pthread_condattr_t *  condition_var_attributes,
              size_t *  condition_var_control_size);
int      px5_pthread_condattr_getname(pthread_condattr_t *  condition_var_attributes, char ** name);
int      px5_pthread_condattr_setcontroladdr(pthread_condattr_t *  condition_var_attributes,
              void * condition_var_control_address, size_t  condition_var_control_size);
int      px5_pthread_condattr_setname(pthread_condattr_t *  condition_var_attributes, char * name);
```

Event Flags APIs

```
int      px5_pthread_event_flags_clear(pthread_event_flags_t *  event_flags_handle);
int      px5_pthread_event_flags_create(pthread_event_flags_t *  event_flags_handle,
              pthread_event_flagsattr_t *  event_flags_attributes);
int      px5_pthread_event_flags_destroy(pthread_event_flags_t *  event_flags_handle);
int      px5_pthread_event_flags_set(pthread_event_flags_t *  event_flags_handle, u_long events_to_set);
int      px5_pthread_event_flags_timedwait(pthread_event_flags_t *  event_flags_handle,
              u_long requested_events, int all_or_any, u_long *  received_events, tick_t  timeout_ticks);
int      px5_pthread_event_flags_trywait(pthread_event_flags_t *  event_flags_handle, u_long requested_events,
              int all_or_any, u_long *  received_events);
int      px5_pthread_event_flags_wait(pthread_event_flags_t *  event_flags_handle, u_long requested_events,
              int all_or_any, u_long *  received_events);
int      px5_pthread_event_flagsattr_destroy(pthread_event_flagsattr_t *  event_flags_attributes);
int      px5_pthread_event_flagsattr_getcontroladdr(pthread_event_flagsattr_t *  event_flags_attributes,
              void ** event_flags_control_address);
int      px5_pthread_event_flagsattr_getcontrolsize(pthread_event_flagsattr_t *  event_flags_attributes,
              size_t * event_flags_control_size);
int      px5_pthread_event_flagsattr_getname(pthread_event_flagsattr_t *  event_flags_attributes, char ** name);
int      px5_pthread_event_flagsattr_init(pthread_event_flagsattr_t *  event_flags_attributes);
int      px5_pthread_event_flagsattr_setcontroladdr(pthread_event_flagsattr_t *  event_flags_attributes,
              void *  event_flags_control_address, size_t  event_flags_control_size);
int      px5_pthread_event_flagsattr_setname(pthread_event_flagsattr_t *  event_flags_attributes, char *  name);
```

Fast Queue APIs

```
int      px5_pthread_fastqueue_create(pthread_fastqueue_t *fastqueue_handle,
              pthread_fastqueueattr_t * fastqueue_attributes, size_t message_size, int max_messages);
int      px5_pthread_fastqueue_destroy(pthread_fastqueue_t * fastqueue_handle);
int      px5_pthread_fastqueue_receive(pthread_fastqueue_t * fastqueue_handle, u_long * message_destination,
              size_t message_size);
int      px5_pthread_fastqueue_send(pthread_fastqueue_t * fastqueue_handle, u_long * message_source,
              size_t message_size);
int      px5_pthread_fastqueue_timedreceive(pthread_fastqueue_t * fastqueue_handle, u_long *
              message_destination,  size_t message_size, tick_t timeout_ticks);
int      px5_pthread_fastqueue_timedsend(pthread_fastqueue_t * fastqueue_handle, u_long * message_source,
              size_t message_size, tick_t  timeout_ticks);
int      px5_pthread_fastqueue_tryreceive(pthread_fastqueue_t * fastqueue_handle, u_long *
              message_destination,  size_t message_size);
int      px5_pthread_fastqueue_trysend(pthread_fastqueue_t * fastqueue_handle, u_long * message_source,
              size_t message_size);
int      px5_pthread_fastqueueattr_destroy(pthread_fastqueueattr_t * fastqueue_attributes);
int      px5_pthread_fastqueueattr_getcontroladdr(pthread_fastqueueattr_t * fastqueue_attributes,
              void ** fastqueue_control_address);
int      px5_pthread_fastqueueattr_getcontrolsize(pthread_fastqueueattr_t * fastqueue_attributes,
              size_t * fastqueue_control_size);
```

Fast Queue APIs (Continues)

```
int      px5_pthread_fastqueueattr_getname(pthread_fastqueueattr_t * fastqueue_attributes, char ** fastqueue_name);
int      px5_pthread_fastqueueattr_getqueueaddr(pthread_fastqueueattr_t* fastqueue_attributes,
              void ** fastqueue_memory_address);
int      px5_pthread_fastqueueattr_getqueuesize(pthread_fastqueueattr_t* fastqeueu_attributes,
              size_t * fastqueue_memory_size);
int      px5_pthread_fastqueueattr_init(pthread_fastqueueattr_t * fastqueue_attributes);
int      px5_pthread_fastqueueattr_setcontroladdr(pthread_fastqueueattr_t * fastqueue_attributes,
              void * fastqueue_control_address, size_t fastqueue_control_size);
int      px5_pthread_fastqueueattr_setname(pthread_fastqueueattr_t * fastqueue_attributes, char * fastqueue_name);
int      px5_pthread_fastqueueattr_setqueueaddr(pthread_fastqueueattr_t* fastqueue_attributes,
              void * fastqueue_memory_address, size_t fastqueue_memory_size);
```

Message Queue APIs

```
int      mq_close(mqd_t  message_queue);
int      mq_getattr(mqd_t  message_queue, struct mq_attr *  queue_attributes);
mqd_t    mq_open(const char *  queue_name, int operation, mode_t  mode, struct mq_attr *  queue_attributes);
ssize_t  mq_receive(mqd_t  message_queue, char *  message, size_t message_size, unsigned int  *message_priority);
int      mq_send(mqd_t  message_queue, const char *  message, size_t  message_size, unsigned int  message_priority);
int      mq_setattr(mqd_t  message_queue, const struct mq_attr *  queue_attributes, struct mq_attr *
              previous_attributes);
ssize_t  mq_timedreceive(mqd_t  message_queue, char *  message, size_t message_size, unsigned int *
              message_priority, const struct px5_timespec *absolute_timeout);
int      mq_timedsend(mqd_t  message_queue, const char *  message, size_t  message_size, unsigned int
              message_priority, const struct px5_timespec *absolute_timeout);
mqd_t    px5_mq_extend_open(const char *  queue_name, int operation, mode_t  mode, struct mq_attr *
              queue_attributes, mq_extendattr_t *  extend_attributes);
int      px5_mq_extendattr_destroy(mq_extendattr_t *  queue_attributes);
int      px5_mq_extendattr_getcontroladdr(mq_extendattr_t *  queue_attributes, void **  queue_control_address);
int      px5_mq_extendattr_getcontrolsize(mq_extendattr_t *  queue_attributes, size_t *  queue_control_size);
int      px5_mq_extendattr_getqueueaddr(mq_extendattr_t *  queue_attributes, void **  queue_memory_address);
int      px5_mq_extendattr_getqueuesize(mq_extendattr_t *  queue_attributes, size_t *  queue_memory_size);
int      px5_mq_extendattr_init(mq_extendattr_t *  queue_attributes);
int      px5_mq_extendattr_setcontroladdr(mq_extendattr_t *  queue_attributes, void *  queue_control_address,
              size_t  queue_contorl_size);
int      px5_mq_extendattr_setqueueaddr(mq_extendattr_t *  queue_attributes, void *  queue_memory_address,
              size_t  queue_memory_size);
mqd_t    px5_mq_extend_open_check_params(const char *  queue_name, int operation, mode_t  mode,
              struct mq_attr *  queue_attributes, mq_extendattr_t *  extend_attributes);
```

Mutex APIs

```
int      pthread_mutex_destroy(pthread_mutex_t *  mutex_handle);
int      pthread_mutex_init(pthread_mutex_t *  mutex_handle, pthread_mutexattr_t *  mutex_attributes);
int      pthread_mutex_lock(pthread_mutex_t *  mutex_handle);
int      pthread_mutex_trylock(pthread_mutex_t *  mutex_handle);
int      pthread_mutex_unlock(pthread_mutex_t *  mutex_handle);
int      pthread_mutexattr_destroy(pthread_mutexattr_t *  mutex_attributes);
int      pthread_mutexattr_getprotocol(pthread_mutexattr_t *  mutex_attributes, int *  protocol);
int      pthread_mutexattr_getpshared(pthread_mutexattr_t *  mutex_attributes, int *  process_sharing_designation);
int      pthread_mutexattr_gettype(pthread_mutexattr_t *  mutex_attributes, int *  type);
int      pthread_mutexattr_init(pthread_mutexattr_t *  mutex_attributes);
int      pthread_mutexattr_setprotocol(pthread_mutexattr_t *  mutex_attributes, int   protocol);
int      pthread_mutexattr_setpshared(pthread_mutexattr_t *  mutex_attributes, int   process_sharing_designation);
int      pthread_mutexattr_settype(pthread_mutexattr_t *  mutex_attributes, int   type);
int      px5_pthread_mutexattr_getcontroladdr(pthread_mutexattr_t *  mutex_attributes,
              void **  mutex_control_address);
int      px5_pthread_mutexattr_getcontrolsize(pthread_mutexattr_t *  mutex_attributes, size_t *  mutex_control_size);
int      px5_pthread_mutexattr_getname(pthread_mutexattr_t *  mutex_attributes, char **  name);
int      px5_pthread_mutexattr_setcontroladdr(pthread_mutexattr_t *  mutex_attributes, void *
              mutex_control_address,  size_t  mutex_control_size);
int      px5_pthread_mutexattr_setname(pthread_mutexattr_t *  mutex_attributes, char *  name);
```

All APIs with leading "px5_" are PX5 RTOS pthread+ extensions.