truetime@control.lth.se

init_truetime.m

```
>> truetime
```

```
>> mex -setup C++
```

segmentdatattCreateTaskttCreatePeriodicTask

ttAnalogInttAnalogOut

Task_DatauK

ttCallBlockSystem

```
function [exectime,data] = ctrl_code(segment, data)

switch segment
  case 1
    y = ttAnalogIn(1);
    data.u = -data.K * y;
    exectime = data.exectime;
  case 2
    ttAnalogOut(1, data.u)
    exectime = -1;
end
```

ttCallBlockSystem

```
function [exectime, data] = PIcode(segment, data)

switch segment,
  case 1,
    inp(1) = ttAnalogIn(1);
    inp(2) = ttAnalogIn(2);
    outp = ttCallBlockSystem(2, inp, 'PI_Controller');
    data.u = outp(1);
    exectime = outp(2);
  case 2,
    ttAnalogOut(1, data.u);
    exectime = -1; % finished
end
```

example_initargument

ttInitKernelttCreatePeriodicTaskPcontroller

```
>> clear functions
```

mymodel

```
>> set_param('mymodel', 'InitFcn', 'clear functions')
```

ttkernel.cppfilenameexample_init.cppS_FUNCTION_NAMEexample_init
init()mdlInitializeSizesttGetInitArginit()cleanup()ttSetUserDatattGetUserData

```
>> make_truetime
```

example_init.cpp

```
>> ttmex example_init.cpp
```

ttmexmexttkernel.cpp

ttInitKernel

```
function simple_init

ttInitKernel('prioFP')

data.K = 2;            % controller proportional gain
data.exectime = 0.1;   % control task execution time
starttime = 0.0;       % control task start time
period = 0.5;          % control task period

ttCreatePeriodicTask('ctrl_task', starttime, period, 'ctrl_code', data)
```

ttkernel.cpp

```
#define S_FUNCTION_NAME filename
#include "ttkernel.cpp"

// insert your code functions here

void init() {
// perform the initialization
}

void cleanup() {
// free dynamic memory allocated in this script
}
```

ttSetKernelParameter

0.011%

ttSetKernelParameter

ttSendMsg

ttSetNetworkParameter

SndRcvSchedule

$$t_{\text{backoff}} = / \times R$$

$R = \text{rand}(0,\, 2^K - 1) K$
$K = 1 K = 2$

```
#define S_FUNCTION_NAME simple_init
#include "ttkernel.cpp"

// Data structure used for the task data
class CtrlData {
public:
  double u;
  double K;
  double exectime;
};

// Code function for the P-controller
double ctrl_code(int segment, void* data) {

  CtrlData *d = (CtrlData*)data;
  double y;

  switch (segment) {
  case 1:
    y = ttAnalogIn(1);
    d->u = - d->K * y;
    return d->exectime;
  case 2:
    ttAnalogOut(1, d->u);
    return FINISHED;
  }
}

// Kernel init function
void init() {

  // Allocate memory for the task
  CtrlData *data = new CtrlData;

  // Store a pointer to the task data so that it can be cleaned up
  ttSetUserData(data);

  data->K = 2.0;
  data->exectime = 0.1;

  ttInitKernel(prioFP);
  ttCreatePeriodicTask("ctrl_task", 0.0, 0.5, ctrl_code, data);
}

// Kernel cleanup function
void cleanup() {

  delete (CtrlData *)ttGetUserData();
}
```

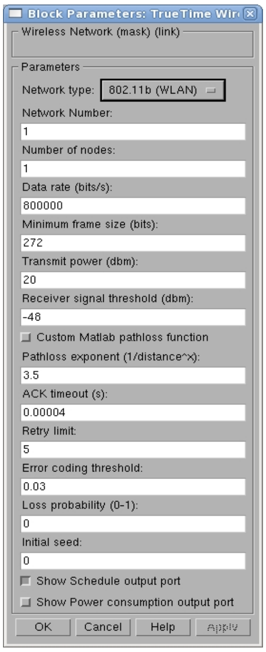
$$t_{\text{idle}} = /,$$

$\times$

$$t_{\text{slot}} = /.$$

n

$(x, y)xy$


Block Parameters: TrueTime Wir
Wireless Network (mask) (link)
Parameters
OK
Cancel
Help
Apply

Network type: 802.11b (WLAN)

Network Number: 1

Number of nodes: 1

Data rate (bits/s): 800000

Minimum frame size (bits): 272

Transmit power (dbm): 20

Receiver signal threshold (dbm): -48

□ Custom Matlab pathloss function

Pathloss exponent (1/distance^x): 3.5

ACK timeout (s): 0.00004

Retry limit: 5

Error coding threshold: 0.03

Loss probability (0-1): 0

Initial seed: 0

□ Show Schedule output port

□ Show Power consumption output port

`xy`

-
-
-
- $\frac{1}{d^a} da$
-

`ttSetNetworkParameter`

802.11802.15.4

`SndRcvSchedule`

$sizeof(header) + sizeof(tail)$

$\frac{1}{d^a} da$

$[0, 1]3$

$\mu\frac{1}{d^a}$

μ

0

$[0, 2^{BE} - 1]$

4

$+1$
$+1$

2

3

5

4

0101
$XX \in Bin(n, p) npnX \in N(np, \sqrt{npq}) q = 1 - pbnb$

$$P(X \leq bn) = \begin{cases} \Phi(\dfrac{bn - np}{\sqrt{npq}}) & \text{if} \quad bn - np > 0 \\ 1 - \Phi(\dfrac{|bn - np|}{\sqrt{npq}}) & \text{if} \quad bn - np \leq 0 \end{cases}$$

Φ

$100n = 1000.1p = 0.1q = 1 - p = 0.95b = 0.05$

$$P(X \leq bn) = 1 - \Phi(\frac{|bn - np|}{\sqrt{npq}}) = 1 - \Phi(\frac{5}{\sqrt{9}}) \approx 0.0478$$

180°

$$P_{receiver} = \frac{1}{d^a} P_{sender}$$

$Pdaxy$

-
-
- xy
-
- xy
-

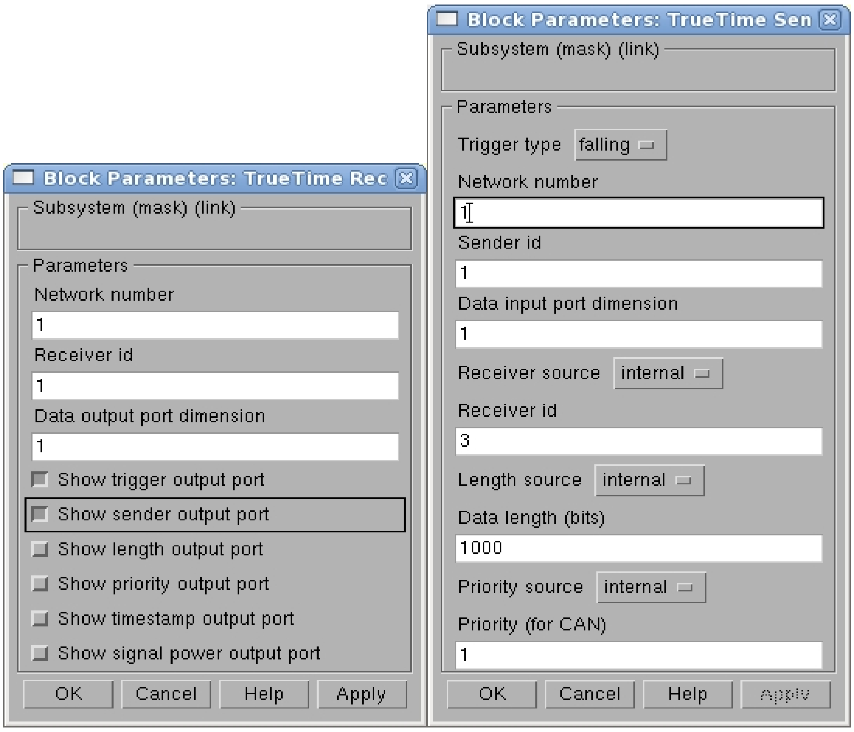
EPP
P

```
function power = rayleigh(transmitPower, node1, x1, y1, node2, x2, y2, time)

% Calculate the exponential pathloss
distance = sqrt((x1 - x2)^2 + (y1 - y2)^2);
power = transmitPower/(distance+1)^3.5;

% Kalman filter to get the relative velocity of the two nodes
velocity = kalman_velocity(node1, x1, y1, node2, x2, y2, time);
% Calculate the rayleigh fading
factor = calculate_rayleigh(node1, node2, velocity, time);

% Add the rayleigh fading to the exponential path loss
power = power * factor;
```


Block Parameters: TrueTime Rec

Subsystem (mask) (link)

Parameters

Network number

1

Receiver id

1

Data output port dimension

1

□ Show trigger output port

□ Show sender output port

□ Show length output port

□ Show priority output port

□ Show timestamp output port

□ Show signal power output port

Block Parameters: TrueTime Sen

Subsystem (mask) (link)

Parameters

Trigger type: falling

Network number

1

Sender id

1

Data input port dimension

1

Receiver source: internal

Receiver id

3

Length source: internal

Data length (bits)

1000

Priority source: internal

Priority (for CAN)

1

ttSendttGetMsg
ttGetMsg

$DIR/examples

simple

threeservos

networked

wireless

AODV

soccer

RUNES_demo

$DIR/examples/simple/matlab

$$G(s) = \frac{1000}{s(s+1)}$$

$$P(k) = K \cdot (\beta r(k) - y(k))$$
$$I(k+1) = I(k) + \frac{Kh}{T_i}(r(k) - y(k))$$
$$D(k) = a_d D(k-1) + b_d(y(k-1) - y(k))$$

servo.mdl

- pidcode1.m
-
-
- controller.mdl121pidcode2.m
- pidcode3.m
- samplercode.mpidcode4.m

```
  function servo_init(mode)

  ttInitKernel(2, 1, 'prioFP'); % nbrOfInputs, nbrOfOutputs, fixed priority

  period = 0.006;
  deadline = period;
  offset = 0.0;
  prio = 1;

  data.K = 0.96;
  ... % more task data

switch mode,
 case 1,
  % IMPLEMENTATION 1: using the built-in support for periodic tasks
  %
  ttCreatePeriodicTask('pid_task',offset,period,prio,'pidcode1',data);

 case 2,
  % IMPLEMENTATION 2: calling Simulink block within code function
  %
  data2.u = 0;
  ttCreatePeriodicTask('pid_task',offset,period,prio,'pidcode2',data2);

 case 3,
  % IMPLEMENTATION 3: sleepUntil and loop back

  data.t = 0;
  ttCreateTask('pid_task',deadline,prio,'pidcode3',data);
  ttCreateJob('pid_task');

 case 4,
  % IMPLEMENTATION 4: sampling in timer handler, triggers task job

  hdl_data.yChan = 2;
  ttCreateInterruptHandler('timer_handler',prio,'samplercode',hdl_data);
  ttCreatePeriodicTimer('timer',offset,period,'timer_handler');
  ttCreateMailbox('Samples',10);
  ttCreateTask('pid_task',deadline,prio,'pidcode4',data);
end
```

$DIR/examples/threeservos/matlab

threeservos.mdl

- ttInitKernelthreeservos_init.m
- ’prioRM’’prioEDF’

$DIR/examples/networked/matlab

networked.mdl

-
- BWshareinterfcode.m’dummy’controller_init
- BWshare

$DIR/examples/wireless

wireless.mdl

-
- power_controller_taskcontroller_init
- 10ttSetKernelParameter

$DIR/examples/AODVnode_init.m
AODVsendcode.m

```
  IF (data message received from application)
     check the routing table for a route to the destination;
     IF (a valid route exists)
        forward data message to next hop on route;
        update expiry time of route entry;
     ELSE
        initiate route discovery by broadcasting RREQ message;
        buffer data message until route has been established;
     END
  ELSE IF (notified of established new route)
     send all buffered data messages to destination
  END
```

AODVrcvcode.m

```
  IF (receiving data message)
     update expiry timer for reverse route entry to source;
     IF (this node is the destination)
        Pass data message to application;
     ELSE
        forward data message to next hop on route;
        update expiry timer of route entry;
     END
  ELSE
     SWITCH message_type
        CASE RREQ:
           IF (first time this RREQ is received)
              enter RREQ in cache;
              create or update route entry to source;
              check the routing table for a route to the destination;
              IF (a route exists)
                 send RREP message back towards source;
              ELSE
                 update and rebroadcast the RREQ;
              END
           END
        CASE RREP:
           check the routing table for a route to the destination;
              IF (no route exists)
                 create route entry to destination;
              ELSE IF (route entry exists but should be updated)
                 update route entry to destination;
              END
              IF (this node is the original source)
                 notify the AODV send task about the new route;
              ELSE IF (route to destination was created or updated)
                 update reverse route entry towards source;
                 propagate RREP to next hop towards source;
              END
        CASE RERR:
              find and invalidate all affected route entries;
              propagate the RERR to all previous hops on the routes;
     END
  END
```

hellocode.mexpcode.m

AODV.mdl

-
- $170.51 \to 3 \to 5 \to 7t = 35t = 10647$routing_table
- initsim.mverbose01
- sentreceived178.00028.50029.0002
- HELLO_INTERVALinitsim.mnode_init.m

$DIR/examples/soccer
xyballmotion.m

RTsys$DIR/kernel/ttkernel.hrtsysrtsysUserDataRTsys

```
class RTsys {
 public:
  double time;              // Current time in simulation

  double* inputs;           // Vector of input port values
  double* outputs;          // Vector of output port values

  Task* running;            // Currently running task

  List* readyQ;   // usertasks and handlers ready for execution, prio-sorted
  List* timeQ;    // usertasks and timers waiting for release, time-sorted

  List* taskList;    // A list containing all created tasks
  List* handlerList;
  List* monitorList;
  List* eventList;

  double (*prioFcn)(Task*); // Priority function
};
```

prioFcnTaskttInitKernel
Task$DIR/kernel/task.hnode$DIR/kernel/linkedlist.h

```
class Task : public Node {
 public:
  char* name;
  int segment;      // the current segment of the code function
  double execTime;  // the remaining execution time of the current segment
  void *data;       // task data (C++ case)
  char* dataMatlab; // name of global variable for task data (Matlab case)

  double (*codeFcn)(int, void*); // Code function (C++ case)
  char* codeFcnMatlab;  // Name of m-file code function (Matlab case)
};
```

exectime
Task$DIR/kernel/usertask.h$DIR/kernel/handler.h

```
class UserTask : public Task {
public:
  double priority;
  double wcExecTime;
  double deadline;
  double absDeadline;
  double release;   // task release time if in timeQ
  double budget;

  int state;  // Task state (IDLE; WAITING; SLEEPING; READY; RUNNING)

  double tempPrio;  // temporarily raised prio value

  List *pending; // list of pending jobs

  InterruptHandler* deadlineORhandler; // deadline overrun handler
  InterruptHandler* exectimeORhandler; // execution-time overrun handler

  int nbrOfUserLogs;  // Number of user-created log entries
  Log* logs[NBRLOGS];

  void (*arrival_hook)(UserTask*);  // hooks
  void (*release_hook)(UserTask*);
  void (*start_hook)(UserTask*);
  void (*suspend_hook)(UserTask*);
  void (*resume_hook)(UserTask*);
  void (*finish_hook)(UserTask*);
};
```

$DIR/kernel/log.hLog

```
class InterruptHandler : public Task {
 public:
  double priority;

  int type;  // {UNUSED, OVERRUN, TIMER, NETWORK, EXTERNAL}

  UserTask *usertask; // if overrun handler to task

  Timer* timer;       // if associated with timer interrupt

  Network* network;   // if associated with network receive interrupt

  Trigger* trigger;   // if associated with external interrupt
  int pending;        // list of pending invocations, if new external
                      // interrupt occurs before the old is served
```

};

$DIR/kernelTimerNetworkTrigger

ttCreateJobttCreatePeriodicTask

-
-
-
-

ttSetX

-
-
-
-

ttAttachDLHandlerttAttachWCETHandler
ttCreateJob
$DIR/kernel/defaulthooks.cpp

runKernel$DIR/kernel/ttkernel.cpp

$DIR/kernel/ttkernel.cpp

FIXED_IN_MINOR_STEP_OFFSET

```
static void mdlInitializeSampleTimes(SimStruct *S) {

   ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
   ssSetOffsetTime(S, 0, FIXED_IN_MINOR_STEP_OFFSET);
}
```

nextHit

```
static void mdlZeroCrossings(SimStruct *S) {

   Store all inputs;
      if (any interrupt input has changed value) {
         nextHit = ssGetT(S);
      }
   ssGetNonsampledZCs(S)[0] = nextHit - ssGetT(S);
}
```

`mdlOutputsmdlOutputsmdlZeroCrossingsmdlZeroCrossings`$< 10^{-15}$
`runKernel()mdlOutputs`

`ttSetXttGetX`
`help commandcommand`

```
double runKernel(void) {

  timeElapsed = rtsys->time - rtsys->prevHit; // time since last invocation
  rtsys->prevHit = rtsys->time;  // update previous invocation time
  nextHit = 0.0;

  while (nextHit == 0.0) {
    // Count down execution time for current task
    // and check if it has finished its execution
    if (there exists a running task) {
      task->execTime -= timeElapsed;
      if (task->execTime == 0.0) {
        task->segment++;
        task->execTime = task->codeFcn(task->segment, task->data);
        if (task->execTime < 0.0) {
          // Negative execution time = task finished
          task->execTime = 0.0;
          task->segment = 0;
          Remove task from readyQ;
          task->finish_hook(task);
          if (job queue is non-empty)
            Release next job and execute release-hook ;
        }
      }
    } // end: counting down execution time of running task

    // Check time queue for possible releases (user tasks or timers)
    for (each task) {
      if ((release time - rtsys->time) == 0.0) {
        Move task to ready queue
      }
    } // end: checking timeQ for releases

    // Determine task with highest priority and make it running task
    newrunning = rtsys->readyQ->getFirst();
    oldrunning = rtsys->running;
    if (oldrunning is being suspended) {
      oldrunning->suspend_hook(oldrunning);
    }
    if (newrunning is being resumed or started) {
      if (newrunning->segment == 0) {
        newrunning->start_hook(newrunning);
      } else {
        newrunning->resume_hook(newrunning);
      }
    } // end: task dispatching

    // Determine next invocation of kernel function
    time1 = remaining execution time of current task;
    time2 = next release time of a task from the time queue
    nextHit = min(time1, time2);
  } // end: loop while nextHit == 0.0
  return nextHit;
}
```

ttInitKernel	
ttGetInitArg (C++ only)	
ttCreateTask	
ttCreatePeriodicTask	
ttCreateLog	
ttCreateHandler	
ttCreateMonitor	
ttCreateEvent	
ttCreateMailbox	
ttCreateSempahore	
ttCreateCBS	
ttNoSchedule	
ttNonPreemptible	
ttAttachTriggerHandler	
ttAttachNetworkHandler	
ttAttachDLHandler	
ttAttachWCETHandler	
ttAttachHook (C++ only)	
ttAttachCBS	
ttAbortSimulation	

ttSetPeriod	
ttSetDeadline	
ttSetPriority	
ttSetWCET	
ttSetData	
ttSetAbsDeadline	
ttSetBudget	
ttSetUserData	
ttGetPeriod	
ttGetDeadline	
ttGetPriority	
ttGetWCET	
ttGetData	
ttGetRelease	
ttGetAbsDeadline	
ttGetBudget	
ttSetUserData	
ttGetUserData	
ttSetCBSParameters	

ttCreateJob	
ttKillJob	
ttEnterMonitor	
ttExitMonitor	
ttWait	
ttNotify	
ttNotifyAll	
ttLogStart	
ttLogStop	
ttLogNow	
ttLogValue	
ttTryPost	
ttTryFetch	
ttPost	
ttFetch	
ttRetrieve	
ttTake	
ttGive	
ttCreateTimer	
ttCreatePeriodicTimer	
ttRemoveTimer	
ttCurrentTime	
ttSleepUntil	
ttSleep	
ttAnalogIn	
ttAnalogOut	
ttSetNextSegment	
ttGetInvoker	
ttCallBlockSystem	
ttSendMsg	
ttUltrasoundPing	
ttGetMsg	
ttDiscardUnsentMessages	
ttSetNetworkParameter	
ttSetKernelParameter	

```
ttAbortSimulation
```

```
ttAbortSimulation()
```

```
for i=1:10
  try
    sim('mymodel')
  catch
  end
end
```

```
value = ttAnalogIn(inpChan)
```

```
double ttAnalogIn(int inpChan)
```

```
 inpChan
```

```
ttAnalogOut
```

```
ttAnalogOut(outpChan, value)
```

```
void ttAnalogOut(int outpChan, double value)
```

outpChan

value

ttAnalogIn

```
ttAttachCBS(taskname, CBSname)
```

```
void ttAttachCBS(char *taskname, char *CBSname)
```

taskname
CBSname

ttCreateCBS, ttCreateTask, ttCreatePeriodicTask, ttInitKernel

```
ttAttachDLHandler(taskname, handlername)
```

```
void ttAttachDLHandler(char* taskname, char* handlername)
```

taskname
handlername

ttCreateHandlerttSetDeadlinettAttachWCETHandler

```
void ttAttachHook(char* taskname, int ID, void (*hook)(UserTask*))
```

taskname

ID ARRIVALRELEASESTARTSUSPENDRESUMEFINISH

hook

ttCreateJob
UserTaskUserTaskTask$DIR/kernel/usertask.h$DIR/kernel/task.h$DIR/kernel/defaulthooks.cpp

```
void myFinishHook(UserTask* task) {

  // Compute execution time (the initial budget of a task job is the WCET)
  double exectime = task->wcExecTime - task->budget;

  // Update estimate
  double lambda = 0.5;
  task->data->Chat = lambda*task->data->Chat + (1.0-lambda)*exectime;

  // Execute default finish hook
  default_finish(task);
}
```

```
ttAttachNetworkHandler(taskname)
ttAttachNetworkHandler(networkNbr, taskname)
```

```
void ttAttachNetworkHandler(char *taskname)
void ttAttachNetworkHandler(int networkNbr, char *taskname)
```

taskname

networkNbr

ttCreateHandler

```
ttAttachTBS(taskname, TBSname)
```

```
void ttAttachTBS(char *taskname, char *TBSname)
```

taskname

TBSname

ttCreateTBS, ttCreateTask, ttCreatePeriodicTask, ttInitKernel

```
ttAttachTriggerHandler(triggerNbr, handlername)
```

```
void ttAttachTriggerHandler(int triggerNbr, char *handlername)
```

triggerNbr
handlername

ttCreateHandler

ttAttachWCETHandler(taskname, handlername)

void ttAttachWCETHandler(char* taskname, char* handlername)

taskname

handlername

ttCreateHandlerttSetWCETttAttachDLHandler

```
outp = ttCallBlockSystem(nbroutp, inp, blockname)
```

```
void ttCallBlockSystem(int nbroutp, double *outp, int nbrinp,
                       double *inp, char *blockname)
```

nbrinp

nbroutp

inp

outp

blockname

```
function [exectime, data] = PIcontroller(segment, data)

switch segment,
  case 1,
    inp(1) = ttAnalogIn(1);
    inp(2) = ttAnalogIn(2);
    outp = ttCallBlockSystem(2,
           inp, 'PI_Controller');
    data.u = outp(1);
    exectime = outp(2);
  case 2,
    ttAnalogOut(1, data.u);
    exectime = -1;
end
```

```
ttCreateCBS(name, Qs, Ts)
ttCreateCBS(name, Qs, Ts, type)
```

```
void ttCreateCBS(char *name, double Qs, double Ts)
void ttCreateCBS(char *name, double Qs, double Ts, int type)
```

name

Qs

Ts

type

ttAttachCBS, ttInitKernel

```
ttCreateEvent(eventname)
ttCreateEvent(eventname, monitorname)
```

```
void ttCreateEvent(char *eventname)
void ttCreateEvent(char *eventname, char *monitorname)
```

eventname

monitorname

ttWaitttNotifyttNotifyAll

```
ttCreateHandler(name, priority, codeFcn)
ttCreateHandler(name, priority, codeFcn, data)
```

```
void ttCreateHandler(char *name, double priority,
                              double (*codeFcn)(int, void*))
void ttCreateHandler(char *name, double priority,
                  double (*codeFcn)(int, void*), void* data)
```

name

priority

codeFcn   codeFcn

data

ttCreateTimerttCreatePeriodicTimerttAttachTriggerHandler
ttAttachNetworkHandlerttAttachDLHandlerttAttachWCETHandler

```
ttCreateJob(taskname)
ttCreateJob(taskname, time)
```

```
void ttCreateJob(char *taskname)
void ttCreateJob(char *taskname, double time)
```

taskname

taskname

ttCreateJobttCreateTaskttCreateJob

ttCreateTaskttKillJob

```
ttCreateLog(logname, variable, size)
ttCreateLog(taskname, logtype, variable, size)
```

```
void ttCreateLog(char* logname, char* variable, int size)
void ttCreateLog(char* taskname, int logtype, char* variable, int size)
```

logname

variable

size

taskname

logtype

logtypettLogStartttLogStopttLogNowttLogValuevariable

```
% In initialization script

% Automatic log of response time (type 1)
ttCreateLog('ctrl_task', 1, 'Responsetime', 100);

% User log for logging of I/O latency
ttCreateLog('mylog', 'IOlatency', 100);
```

```
% Code function
function [exectime, data] = ctrl(seg, data)

switch seg,
  case 1,
    ttLogStart('mylog');     % Start I/O logging in user log
    y = ttAnalogIn(1);       % Input
    data.u = calculateOutput(y);
    exectime = 0.003;
  case 2,
    ttLogStop('mylog');      % Stop and write log entry in user log
    ttAnalogOut(1, data.u);  % Output
    exectime = -1;
end
```

ttLogNowttLogStartttLogStop

```
ttCreateMailbox(mailboxname)
ttCreateMailbox(mailboxname, maxsize)
```

```
void ttCreateMailbox(char *mailboxname)
void ttCreateMailbox(char *mailboxname, int maxsize)
```

mailboxname

maxsize

maxsize

ttTryPostttTryFetchttPostttFetchttRetrieve

```
ttCreateMonitor(name, display)
```

```
void ttCreateMonitor(char *name, bool display)
```

name
display

ttCreateEvent

ttEnterMonitorttExitMonitorttCreateEventttWaitttNotifyttNotifyAll

```
ttCreatePeriodicTask(name, starttime, period, codeFcn)
ttCreatePeriodicTask(name, starttime, period, codeFcn, data)
```

```
void ttCreatePeriodicTask(char* name, double starttime, double period,
       double (*codeFcn)(int, void*))
void ttCreatePeriodicTask(char* name, double starttime, double period,
       double (*codeFcn)(int, void*), void* data)
```

name

starttime

period

codeFcn   codeFcn

data

$DIR/examples/simple

ttCreateTaskttSetX

```
ttCreatePeriodicTimer(timername, period, handlername)
ttCreatePeriodicTimer(timername, offset, period, handlername)
```

```
void ttCreatePeriodicTimer(char *timername, double period, char *handlername)
void ttCreatePeriodicTimer(char *timername, double offset, double period,
                           char *handlername)
```

timername

offset

period

handlername

ttCreateInterruptHandlerttCreateTimerttRemoveTimer

```
ttCreateSemaphore(name, initval)
ttCreateSemaphore(name, initval, maxval)
```

```
void ttCreateSemaphore(char *name, int initval)
void ttCreateSemaphore(char *name, int initval, int maxval)
```

name

initval

maxval

ttGivemaxvalmaxval

ttTakettGive

```
ttCreateTask(name, deadline, codeFcn)
ttCreateTask(name, deadline, codeFcn, data)
```

```
void ttCreateTask(char* name, double deadline,
                  double (*codeFcn)(int, void*))
void ttCreateTask(char *name, double deadline,
                  double (*codeFcn)(int, void*), void* data)
```

name

deadline

codeFcn codeFcn

data

ttCreatePeriodicTaskttCreateJobttSetX

```
ttCreateTimer(timername, time, handlername)
```

```
void ttCreateTimer(char *timername, double time, char *handlername)
```

```
 timername
 time
 handlername
```

```
ttCreateInterruptHandlerttCreatePeriodicTimerttRemoveTimer
```

```
time = ttCurrentTime
time = ttCurrentTime(newTime)
```

```
double ttCurrentTime()
double ttCurrentTime(double newTime)
```

`newTime`

```
nbr = ttDiscardUnsentMessages
nbr = ttDiscardUnsentMessages(network)
```

```
int ttDiscardUnsentMessages()
int ttDiscardUnsentMessages(int network)
```

`network`

ttEnterMonitor(monitorname)

void ttEnterMonitor(char *monitorname)

monitorname

ttEnterMonitorttEnterMonitor

```
function [exectime, data] = ctrl(seg, data)
switch seg,
  case 1,
    ttEnterMonitor('mutex');
    exectime = 0;
  case 2,
    if some_condition_not_fulfilled
      ttWait('condvar');
      ttSetNextSegment(2);
      exectime = 0;
    else
      criticalOperation;
      exectime = 0.005;
    end
  case 3,
    ttExitMonitor('mutex');
    exectime = -1;
end
```

ttCreateMonitorttExitMonitorttCreateEventttWaitttNotifyttNotifyAll ttExitMonitor(monitorname)

```
void ttExitMonitor(char *monitorname)
```

monitorname

ttExitMonitor

ttEnterMonitor

ttCreateMonitorttEnterMonitor

ttFetch(mailboxname)

void ttFetch(char *mailboxname)

mailboxname

ttRetrieve

```
function [exectime, data] = ctrl(seg, data)
switch seg,
  case 1,
    ttFetch('mailbox');            % wait for a message
    exectime = 0;
  case 2,
    msg = ttRetrieve('mailbox');   % read the actual message
    doStuff;
    exectime = 0.005;
  case 3,
    exectime = -1;
end
```

ttCreateMailboxttTryPostttTryFetchttPostttRetrieve

```
data = ttGetData(taskname)
```

```
void *ttGetData(char *taskname)
```

taskname

ttSetData

ttSetData, ttCreateTask, ttCreatePeriodicTask

```
mxArray *ttGetInitArg()
```

```
invoker = ttGetInvoker
```

```
char *ttGetInvoker()
```

```
[msg, signalPower] = ttGetMsg
[msg, signalPower] = ttGetMsg(network)
```

```
void *ttGetMsg()
void *ttGetMsg(int network)
void *ttGetMsg(int network, double *signalPower)
```

network

signalPower

```
% Task that waits for and reads messages
function [exectime, data] = receiver(seg, data)
switch seg,
  case 1,
    ttWait('message');
    exectime = 0;
  case 2,
    msg = ttGetMsg;
    disp('I got a message!');
    exectime = 0.001;
  case 3,
    ttSetNextSegment(1); % loop back and wait for new message
    exectime = 0;
end
% Interrupt handler that is called by the network interface
function [exectime, data] = msgRcvhandler(seg, data)
ttNotifyAll('message');
exectime = -1;
```

ttAttachNetworkHandler, ttSendMsg

```
value = ttGetX
value = ttGetX(taskname)
```

```
double ttGetX()
double ttGetX(char *taskname)
```

taskname

- ttGetArrival
- ttGetRelease
- ttGetDeadline
- ttGetAbsDeadline
- ttGetPriority
- ttGetPeriod
- ttGetWCET
- ttGetBudget

ttGetXtaskname

ttSetX

```
ttGive(semname)
```

```
void ttGive(char *semname)
```

semname

```
if (value < maxval) {
  value++;
  if (value <= 0) {
    release the first task from the semaphore queue;
  }
}
```

ttCreateSemaphorettTake

```
ttInitKernel(prioFcn)
ttInitKernel(prioFcn, contextSwitchOverhead)
```

```
void ttInitKernel(double (*prioFcn)(UserTask*))
void ttInitKernel(double (*prioFcn)(UserTask*), double contextSwitchOverhead)
```

prioFcn

contextSwitchOverhead

'prioFP''prioDM''prioEDF'

prioFPprioDMprioEDF

ttNoSchedule('CShandler')

```
/* Priority function for deadline-monotonic scheduling */
double prioDM(UserTask* t) {
  return t->deadline;
}

/* Priority function for earliest-deadline-first scheduling,
   with support for constant bandwidth servers */
double prioEDF(UserTask* t) {
  if (t->cbs) {
    // The task is associated with a CBS: inherit the deadline of the CBS
    return t->cbs->ds;
  } else {
    // No CBS: return the absolute deadline of the task
    return t->absDeadline;
  }
}
```

```
ttKillJob(taskname)
```

```
void ttKillJob(char *taskname)
```

taskname

ttCreateJob

ttLogNow(logname)

```
void ttLogNow(char *logname)
```

logname

ttCreateLogttLogStartttLogStopttLogValue ttLogStart(logname)

void ttLogStart(int logname)

logID

ttLogStop

ttCreateLogttLogStartttLogStop

ttCreateLogttLogStopttLogNowttLogValue

ttLogStop(logname)

void ttLogStop(int logname)

logname

ttLogStart

ttCreateLogttLogStartttLogStop

ttCreateLogttLogStartttLogNowttLogValue

```
ttLogValue(logname, value)
```

```
void ttLogValue(char *logname, double value)
```

logname

value

ttCreateLogttLogStartttLogStopttLogNow

```
ttNonPreemptible(taskname)
```

```
void ttNonPreemptible(char* taskname)
```

```
 taskname
```

```
ttNoSchedule(name)
```

```
void ttNoSchedule(char* name)
```

`name`

`'CShandler'`

ttNotify(eventname)

void ttNotify(char *eventname)

eventname

ttNotifyttEnterMonitor-ttExitMonitor

ttCreateEventttWaitttNotifyAll

`ttNotifyAll(eventname)`

```
void ttNotifyAll(char *eventname)
```

`eventname`

`ttNotifyAll` `ttEnterMonitor` `ttExitMonitor`

`ttCreateEvent` `ttWait` `ttNotify`

```
ttPost(mailboxname, msg)
```

```
void ttPost(char *mailboxname, void* msg)
```

mailboxname

msg

ttCreateMailboxttTryPostttTryFetchttFetchttRetrieve

```
ttRemoveTimer(timername)
```

```
void ttRemoveTimer(char *timername)
```

timername

ttCreateTimerttCreatePeriodicTimer

ttFetch

msg = ttRetrieve(mailboxname)

void* ttRetrieve(char *mailboxname)

mailboxname

ttFetch

```
function [exectime, data] = ctrl(seg, data)
switch seg,
  case 1,
    ttFetch('mailbox');            % wait for a message
    exectime = 0;
  case 2,
    msg = ttRetrieve('mailbox');   % read the actual message
    doStuff;
    exectime = 0.005;
  case 3,
    exectime = -1;
end
```

ttTryPostttTryFetchttPostttFetch

```
ttSendMsg(receiver, data, length)
ttSendMsg(receiver, data, length, priority)
ttSendMsg([network receiver], data, length)
ttSendMsg([network receiver], data, length, priority)
```

```
void ttSendMsg(int receiver, void *data, int length)
void ttSendMsg(int receiver, void *data, int length, int priority)
void ttSendMsg(int network, int receiver, void *data, int length)
void ttSendMsg(int network, int receiver, void *data, int length, int priority)
```

network

receiver

data

length

priority

ttGetMsg

```
ttSetCBSParameters(cbsname, Qs, Ts)
```

```
void ttSetCBSParameters(char *cbsname, double Qs, double Ts)
```

cbsname

Qs

Ts

ttCreateCBS

```
ttSetData(taskname, data)
```

taskname

data

ttGetData

ttSetDatattGetData

ttGetData, ttCreateTask, ttCreatePeriodicTask

```
ttSetKernelParameter(parameter, value)
```

```
void ttSetKernelParameter(char* parameter, double value)
```

parameter
value

- cpuscaling
- energyconsumption

cpuscalingenergyconsumption

```
ttSetNetworkParameter(parameter, value)
ttSetNetworkParameter(networkNbr, parameter, value)
```

```
void ttSetNetworkParameter(char* parameter, double value)
void ttSetNetworkParameter(int networkNbr, char* parameter, double value)
```

parameter

value

networkNbr

'transmitpower'

'predelay'

'postdelay'

ttSetNextSegment(segment)

```
void ttSetNextSegment(int segment)
```

segment

ttWait

```
ttSetX(value)
ttSetX(value, taskname)
```

```
void ttSetX(double value)
void ttSetX(double value, char *taskname)
```

`value`

`taskname`

- `ttSetDeadline`
- `ttSetAbsDeadline`
- `ttSetPriority`
- `ttSetPeriod`
- `ttSetWCET`
- `ttSetBudget`

`ttSetXttCreateTaskttCreatePeriodicTasktaskname`

$h_1h_12h_13h_1h_1+\tau h_1+h_2h_1+2h_2h_1+3h_2h_2$

`ttCreateTaskttCreatePeriodicTaskttGetX`

```
ttSleep(duration)
```

```
void ttSleep(double duration)
```

```
 duration
```

```
ttSleepUntil(duration + ttCurrentTime())
```

```
ttSleepUntil
```

```
ttSleepUntil(time)
```

```
void ttSleepUntil(double time)
```

time

ttSleep

ttTake(semname)

```
void ttTake(char *semname)
```

semname

```
value--;
if (value < 0) {
  suspend the task and put it in the semaphore queue;
}
```

```
function [exectime, data] = producer_code(seg, data)
switch seg,
  case 1,
    produce_data;
    exectime = 0.020;
  case 2,
    ttTake('sem'); % wait until the consumer task is ready
    exectime = 0;
  case 3,
    send_data_to_consumer;
    exectime = 0.005;
  case 4,
    exectime = -1;
end
```

ttCreateSemaphorettGive

```
msg = ttTryFetch(mailboxname)
```

```
void* ttTryFetch(char* mailboxname)
```

mailboxname

ttCreateMailboxttTryPostttPostttFetchttRetrieve

```
ok = ttTryPost(mailboxname, msg)
```

```
bool ttTryPost(char* mailboxname, void* msg)
```

mailboxname

msg

truefalse

ttCreateMailboxttTryFetchttPostttFetchttRetrieve

ttWait(eventname)

void ttWait(char *eventname)

eventname

ttEnterMonitor-ttExitMonitor

```
function [exectime, data] = ctrl(seg, data)
switch seg,
 case 1,
   ttWait('Event1');
   exectime = 0.0;
 case 2,
   performCalculations;
   exectime = 0.001;
 case 3,
   ttSetNextSegment(1); % loop and wait for new event
   exectime = 0.0;
end
```

ttEnterMonitorttCreateEventttNotifyttNotifyAll