#include "Thread.h"

#include "ThreadController.h"

ThreadController::ThreadController(unsigned long \_interval): Thread(){

cached\_size = 0;

clear();

setInterval(\_interval);

#ifdef USE\_THREAD\_NAMES

// Overrides name

ThreadName = "ThreadController ";

ThreadName = ThreadName + ThreadID;

#endif

}

/\*

ThreadController run() (cool stuf)

\*/

void ThreadController::run(){

// Run this thread before

if(\_onRun != NULL)

\_onRun();

unsigned long time = millis();

int checks = 0;

for(int i = 0; i < MAX\_THREADS && checks <= cached\_size; i++){

// Object exists? Is enabled? Timeout exceeded?

if(thread[i]){

checks++;

if(thread[i]->shouldRun(time)){

thread[i]->run();

}

}

}

// ThreadController extends Thread, so we should flag as runned thread

runned();

}

/\*

List controller (boring part)

\*/

bool ThreadController::add(Thread\* \_thread){

// Check if the Thread already exists on the array

for(int i = 0; i < MAX\_THREADS; i++){

if(thread[i] != NULL && thread[i]->ThreadID == \_thread->ThreadID)

return true;

}

// Find an empty slot

for(int i = 0; i < MAX\_THREADS; i++){

if(!thread[i]){

// Found a empty slot, now add Thread

thread[i] = \_thread;

cached\_size++;

return true;

}

}

// Array is full

return false;

}

void ThreadController::remove(int id){

// Find Threads with the id, and removes

bool found = false;

for(int i = 0; i < MAX\_THREADS; i++){

if(thread[i]->ThreadID == id){

thread[i] = NULL;

cached\_size--;

return;

}

}

}

void ThreadController::remove(Thread\* \_thread){

remove(\_thread->ThreadID);

}

void ThreadController::clear(){

for(int i = 0; i < MAX\_THREADS; i++){

thread[i] = NULL;

}

cached\_size = 0;

}

int ThreadController::size(bool cached){

if(cached)

return cached\_size;

int size = 0;

for(int i = 0; i < MAX\_THREADS; i++){

if(thread[i])

size++;

}

cached\_size = size;

return cached\_size;

}

Thread\* ThreadController::get(int index){

int pos = -1;

for(int i = 0; i < MAX\_THREADS; i++){

if(thread[i] != NULL){

pos++;

if(pos == index)

return thread[i];

}

}

return NULL;

}