User Interface (ui) for the Arduino

Garrett Berg, Cloudform Design, garrett@cloudformdesign.com

Introduction

There was once upon a time when I tried to code my own user interface. It went a little something like this:

```
void mythread(){
  static int place = 0;
  if(place) == 0){
    do something...
    place = 1;
  }
  if(place == 1)
     something else;
     place = 2;
  ... etc.
}

void loop(){
  mythread();
}
```

This way of doing things had serious flaws, too many to go into.

Then I came across the protothreads library (included in this folder as **pt.h**) and everything was better... or was it?

I still wanted to be able to actually *interface* with my functions and variables through the command terminal.

So, to do this I wrote **threads.h** and **ui.h**. With these two libraries it is now possible to:

- Create threads easily — a thread is a function that will be called every loop, who's status can be observed externally, and who can be killed by the implemented task manager. Threads are easy to create and much easier to maintain than doing similar things any other way.
- Access variables easily : after exposing a variable, you can access it with "v myvar" on the serial terminal, or set with "v myvar value"

- Call functions: same as exposing variables.
- Functions and variables take up zero SRAM, and threads only take up 4 bytes each! This allows you to easily construct a very large user interface through exposed functions and variables without worrying about SRAM usage.

Tutorial

Please see UserGuide_threading.h for the functions that you can call. See the example **ui_led_example4** for code examples.

The best way to see the power of the threading module is to see it in action. Do the following:

- Load up **ui_led_example4**
- Program it to your arduino
- Set your baud rate = 57600 and set it to send both NR and CR characters on each enter.
- Play with it!
    - type **“?”** to get info on commands you can type.
    - "**v sub_time 500**" — you will see the led’s blink much faster (500ms is being taken off both periods)
    - "**v sub_time**" — get the value of sub_time in hex
    - "**v sub_time 0**" — set it back to 0 (slow blinking)
    - "**k led1**" — will make it so that only 1 thread is blinking (blinks at period of 900ms)
    - **“k led2”** -- everything will be killed (no blinking)
    - **“reinit”** -- calls the reinit function, everything will be set to default values!
- Check out the code and read the documentation to see how things are working!
- Check out the other documentation:
    - **UserGuide_errorhandling** is how you can generate error messages and do error handling. It also has logging.
    - **UserGuide_threading** is a useful reference on how to create your threads
    - **UserGuide_pt** The protothread library is how you create your threads — it would be helpful to read this.

Serial Port

Threading operates as simply as it possibly can. You can access either functions or variables through it's interface. In summary:

- The following terminal commands are supported:
    - **? ::** print all options for threads, variables and function calls.
    - **t (exposed thread) [input] ::** calls an exposed thread, sends it input.
    - **v** ::VARNAME" — Gives you the value of variable VARNAME in hex that you have exposed
    - **k [process name] ::** kills the indicated process name (integer or exposed name)
    - **(exposed function) [input] ::** calls a function that you have exposed. Sends it input

Overview

The ui module is almost exactly the same as the threading module. The only difference is that you can also expose variables and functions. After you have exposed things, you will be able to see them (by typing ?) as well as interract with them (threads can be killed and scheduled, functions can be called with inputs, variables can be viewed and set)

- **setup_ui(Mem_Amount) ::**
    - Must be called after expose_threads, expose_functions, expose_variables.
    - If one of these is not exposed, **no_NAME()** must be called. i.e. if you are not using any functions, call **no_functions()**
    - Mem_Amount == the maximum amount of dynamic memory that your program will use. See **UserGuide_ReMem** for more details
- **expose_functions(...) ::**
    - exactly the same as exposing threads (even the funciton type is the same), except you must wrap each value in **UI_F** instead of **TH_T**. See **expose_threads** in **UserGuide_threading**
- **expose_variables(...) ::**
    - Exposing variables is slightly different. This is because the compiler can't store this type directly into PROGMEM. To expose variables, you must first create pointers to them using **UI_V(pointer, value)** and then expose them, wrapping their pointers with **UI_VA**:

```
// variables we will be exposing
uint8_t myint;
float myfloat;

// This creates pointers v1 and v2 to each variable
UI_V(v1, myint);
UI_V(v2, myfloat);

// put pointers into exposed array
expose_variables(UI_VA(v1), UI_VA(v2));
```

- **expose_thread_names(...), expose_function_names(...), expose_variable_names(...) ::**
  - Used to access via terminal
  - If you use these, make sure to name every exposed thread/function/variable. For instance, if you have two threads then include exactly two names (order is important)
  - Must be exposed in a similar way to variables:
  - Must call associated **set_NAME_names()** in the setup function. See below.

```
UI_STR(tn1, mythread);
expose_thread_names(tn1);

UI_STR(fn1, myfunc1);
UI_STR(fn2, myfunc2);
expose_function_names(fn1, fn2);

UI_STR(vn1, myvar);
expose_variable_names(vn1);
```

- **expose_default_thread_names()**
  - Only use if you are not exposing ANY threads
  - Allows you to see the default thread **ui** — or the user interface, and kill it with **kill ui**
  - This function is not all that useful
- **expose_default_function_names() ::**
  - Only use if you are not exposing ANY functions
  - Allows you to see and call the default functions by name: **?, t, v, k**
  - Useful if you are not using any functions
- **set_thread_names(), set_function_names(), set_variable_names() ::**

- Must be called in the setup function if names are exposed.
- Can also call **set_all_names()** to do all of them at once.
