# ARDTOS QuickStart Manual ver. 0.0.1

William R Cooke

Nov 11, 2016

# Contents

# Chapter 1

# Quick Start

This chapter is intended to get you started quickly without a lot of study. We will install the ardtos library and then build a simple application with it. Along the way we will look at the principals involved so that you will understand how to use the ardtos library for your own applications

## 1.1 Introduction

the aRdTOS libarary is a small and simple, efficient, and easy to use *Real Time OperatingSystem (RTOS.)* An operating system is a manager of computing resources, and a real time operating system is one that concentrates on managing time: making things happen at specific times.

There are a lot of RTOSs available, with a wide variety of features and capabilies. Mosf are intended for professional programmers on larger platforms than Arduino. I developed aRdTOS because I felt it was time Arduino users got to experience the benefits that an RTOS brings.

Most Arduino users will soon run into a problem of writing their sketch so that certain things happen at certain times. For instance, say you want your sketch to read a distance sensor ten times a second, while simultaneously watching for input from the serial port. That can often be pretty tricky, especially for people with not much experience. But using an RTOS, especially aRdTOS, it can be quite easy and straightforward.

## 1.2 Installation

Download the library .zip file and install it using the library manager. Restart the IDE.

## 1.3 Creating a Simple aRdTOS Sketch

There are four parts to creating an aRdTOS sketch: create the task functions, create the two task lists, initialize the Operating System, and turn control over to the operating system.

The tasks are simple functions that take no parameters and return nothing. There are two types: regular tasks and interrupt tasks.

After installing the library and restarting, create a new sketch and then select "Sketch/include library/aRdTOS" to include the ardtos library. Delete all the code shown in the new sketch and either ype in or copy and paste the sketch below. Compile and download the sketch and open the serial monitor. The on-board LED on the Arduino should blink off and on each second, and about every two seconds a message should print in the serial monitor.

```
#include <OS.h>

// This is an ITASK
void i_blink()
{
  static uint8_t state = 0;
  if(state == 0)
  {
    state = 1;
    digitalWrite(13, HIGH);
  }
  else
  {
    state = 0;
    digitalWrite(13, LOW);
  }
}

// This is a regular task
void message()
{
  Serial.print("millis = ");
  Serial.println(millis());
}

// Create the regular task list and itask list
TASKLIST
TASK(message, 0, 2000, 1)
ENDLIST

ITASKLIST
ITASK(i_blink, 0, 1000, 1)
```

```
ENDLIST

void setup()
{
  Serial.begin(9600);
  pinMode(13, OUTPUT);

  OS_Setup(); // Must include to initialize ardtos
}

void loop()
{
  Serial.println("Starting OS");

  OS_Run();  // Turn control over to ardtos
}
```

## 1.4 About the Sketch

The above sketch is very simple, defining one itask and one regular task. Let's take a quick look at the parts.

The line

```
#include <OS.h>
```

tells the compiler to use the ardtos header file and library. It will be included in every ardtos sketch.

The function i_blink() will blink an led attached to pin 13 (the built in LED.) It will be used as an ITask so it is time critical. Notice that it is very short and simple so that it runs fast. The function message() will print the elapsed number of milliseconds on the serial monitor. It will run as a regular task and so is not time critical.

After we have created the above functions that will be used as Tasks and ITasks, we have to create the two task lists. You MUST create both lists and neither can be empty. If you don't need one type of task, you must create an empty function that does nothing and create a task from that. It can be set to inactive if desired, but it must be present in the task list. The two types of lists are currently identical, but that may change in the future.

A task definition, of either type, consists of four parts: the function name, the starting time, they cycle time, and the active/inactive indicator.

The function name is simply the name of the function to use as this task. Notice that it has no parentheses, just the bare name. (It creates a pointer to the function, if you care.)

The second item is an integer that represents the number of ticks to wait before running the task the first time. This provides a way to have the tasks start at offset times from one another.

The third item is the cycle time, or the number of ticks to wait between running the task each time. In the i_blink() function we want the LED to blink on for one second and off for one second, so we set this value to 1000 (about 1000 milliseconds.) For the message task, we want it to print a message about every two seconds, so we set this value to 2000. A tick is a little more than one millisecond. Whe you run the sketch and look at the serial monitor, you will see that there are about 2048 millis (milliseconds) for each 2000 ticks when the message prints.

The last item is either 0 or 1. If it is zero, the task is created but will be inactive. In other words, it will not run. There are functions that can be used to activate it when needed. If the value is 1, the task will begin running immediately. Keep in mind that if no tasks are active, nothing will work. There must be at least one active task.

The setup() function works as normal. We set the LED pin to an output and turn on the serial communication. Then, we have the required call for ardtos, OS_Setup(). That line initializes the operating system (OS) and sets up all the tasks that we defined in the two task lists.

The loop() function is a little bit special. In a normal Arduino sketch, the code inside the loop() function runs repeatedly, starting over each time it finishes. ardtos replaces that functionality with its own version. The code within your loop() function will execute only once, down to the required OS_Run() line. At that line, operation is turned over to ardtos, which will run all your tasks in a loop similar to what the loop() function does. But it adds the ability to schedule all your tasks for timing purposes, activating and deactivating each task and running them only as the cycle time dictates.

## 1.5 How it Works

aRdTOS tries to stay out of your way as much as possible. It uses as few resources as possible, leaving the maximum for your program. Instead of using a timer for itself, ardtos "piggybacks" on the timer used for millis(). The only thing lost is one channel of PWM. It is also designed to be as small and simple as possible, leaving most of the memory for your code.

When running, the OS cycles through the list of regular tasks, checking to see if they are ready to run. Ready to run means the task is activated and it has reached its elapsed time for its cycle. If both of those are true, the function is called and will run until it returns. Then the next task is examined until the end of the task list, at which time it goes back to the start and repeats. This is very similar to what the loop() function normally does, but with task scheduling added.

The ITasks are run differently. Each time the timer interrupt happens, once every millisecond or so, the list of ITasks is started from the first one. The same process is used as with regular tasks to decide if each task should run. The entire list is checked and all tasks that are ready to run and their timer has elapsed will run to completion. Once the entire list is finished, it does not restart. All the

timers for all tasks are decreased by one, and the program returns to running regular tasks. Keep in mind that your regular tasks will be interrupted by the ITasks so that they can run every millisecond.

## 1.6 Operating System Functions

There are a number of functions that are part of aRdTOS to control tasks. They will be described briefly below. One thing that is common to many of them is a task number. The tasks are numbered separately by type. Each task list starts at zero and increases upward. The first task listed in TASKLIST will be regular task 0. The first task listed in ITASKLIST will be itask 0. The third task in TASKLIST will be regular task 2, and the fourth one listed in ITASKLIST will be itask 3.

### 1.6.1 OS_Deactivate

void OS_Deactivate() will set the currently running task to inactive. It applies to either type of task. To run again, some other task will have to set the task back to active.

### 1.6.2 OS_DeactivateTask(task)

void OS_DeactivateTask(task) will deactivate the regular task whose task number is given by "task."

### 1.6.3 OS_DeactivateITask(itask)

void OS_DeactivateITask(itask) will deactivate the itask whose task number is given by "itask."

### 1.6.4 OS_ActivateTask(task)

void OS_ActivateTask(task) will activate the regular task whose task number is given by "task."

### 1.6.5 OS_ActivateITask(itask)

void OS_ActivateITask(task) will activate the itask whose task number is given by "itask."