

## Radio

## Timers Time and Timers

In this lesson, you will learn how Timers differ from simple `wait1Msec` commands, and how ROBOTC handles them.

The rules for the game state that you have only two minutes from the starting signal to control your robot, and then your robot must *stop* responding to commands or face disqualification. Your program does not currently enforce this rule, and so your robot is *not yet tournament legal!* Let's fix that.

At first glance, it seems that since we already know how to use the `wait1Msec (time) ;` command to make the robot go straight or turn for a certain amount of time, we should be able to follow the same pattern to make the robot do any other behavior for a length of time.

But, try as we might, we can't seem to find the right place to put the command that will make it work! Below is the Radio Control code from the end of the Control Mapping chapter. For simplicity, it is shown without any scaling on the remote control commands (which is usually customized based on driver preference).

```

1 task main()
2 {
3
4     bIfiAutonomousMode = false;
5     bMotorReflected[port2] = 1;
6
7     while(1 == 1)
8     {
9         motor[port3] = vexRT[Ch3];
10        motor[port2] = vexRT[Ch2];
11    }
12 }

```

**Location A**

Putting the `wait1Msec` command here just makes the robot wait a few seconds before letting the remote control behavior start (it still runs forever after that).

**Location B**

Since the (condition) controls how long the loop runs, this seems like it might be a good place, but putting a `wait1Msec` command here confuses the compiler and causes an error.

**Location C**

This makes the code look like the familiar forward and turning commands. However, putting the `wait1Msec` here causes the robot to lock in single power commands for your set amount of time, rather than making the behavior itself run for that long.

**Location D**

Clever, but since the `while ()` loop above is infinite, the program won't ever reach this code.

## Radio

### Timers Time and Timers (cont.)

There is no right place to put a `wait1Msec` command to get the robot to perform a complex behavior for any amount of time. `wait1Msec` does not mean “continue the last behavior for this many milliseconds.” Rather, it means, “go to **sleep** for this many milliseconds.”

Example from *Sample Programs > Moving Forward*:

```
task main()
{
    bMotorReflected[port2]= 1;
    motor[port3] = 127;
    motor[port2] = 127;
    wait1Msec(1000);
}
```

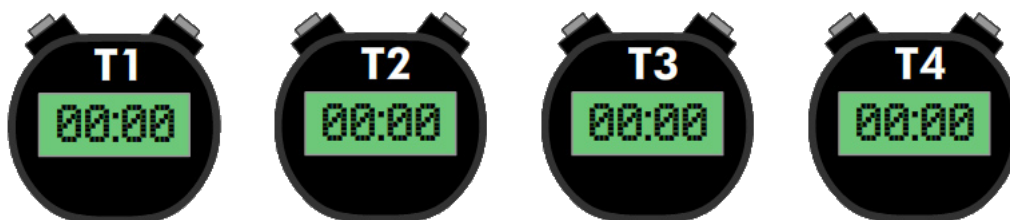
#### Time (not Timer) behavior

Turns on both motors, then goes to sleep for 1 second with the motors still running. The robot will blindly move forward for 1 second.

This method can not be extended to work for behaviors where the robot has to run other commands during the waiting period.

In the **Moving Forward** example above, you’ve really told the robot to put its foot on the gas pedal, go to sleep, and hit the brakes when it wakes up again. That works in the case of just moving forward, but it doesn’t work when the robot needs to listen and respond to radio commands during that time. Instead of sleeping, we’ll keep the robot awake and attentive, using a Timer (rather than just Time) to decide when to stop.

Your robot is equipped with four **Timers**, T1 through T4, which you can think of as Time Sensors, or if you prefer, programmable stopwatches.



Using the Timers is pretty straightforward: you reset a timer with the `ClearTimer()` command, and it immediately starts counting time.

Then, when you want to find out how long it’s been since then, you just use the `time1[TimerName]` command. It gives you the value of the timer in the same way that the `vexRT[ChannelNumber]` command gives you the value of a joystick or button input. The `time1[TimerName]` command gives you the amount of time since the last reset, in milliseconds.

```
ClearTimer(TimerName);

while(time1[TimerName] < 5000)
    ...
```

#### Reminder

Timers should be reset when you are ready to start counting.

`time1[TimerName]` represents the timer value in milliseconds since the last reset. It is shown here being used to make a while loop run until 5 seconds have elapsed.

## Radio

### Timers Time and Timers (cont.)

#### Helpful Hints

The `Timer` function has limitations. The default `time1[TimerName]` can only count to around 30 seconds (see description below). Since we need to count to 2 minutes, we will need to use the `time10[TimerName]` version instead. Keep this in mind as we move on to the next lesson, where you will use the `Timer` to control the length of the radio controlled behavior.

Timer Read Command	Units	Maximum Length of time	Example
<code>time1[TimerName]</code>	milliseconds (ms) = 1/1000ths of a second	32767ms $\approx$ 32.8 seconds	30000 = 30 seconds
<code>time10[TimerName]</code>	centiseconds (cs) = 1/100ths of a second	32767cs $\approx$ 328 seconds $\approx$ 5 1/2 minutes	12000 = 120 seconds = 2 minutes
<code>time100[TimerName]</code>	deciseconds (ds) = 1/10ths of a second	32767ds $\approx$ 3277 seconds $\approx$ 54 1/2 minutes	18000 = 1800 seconds = 30 minutes

#### Why time1 can only count to 30 seconds:

No `Timer` value can be read if the number value it would produce is more than about 30000 (specifically, 32767 or  $2^{15-1}$ ). This is because 32767 is the largest number ROBOTC can fit in a standard integer variable. Other larger variable types do exist, but they require special handling.

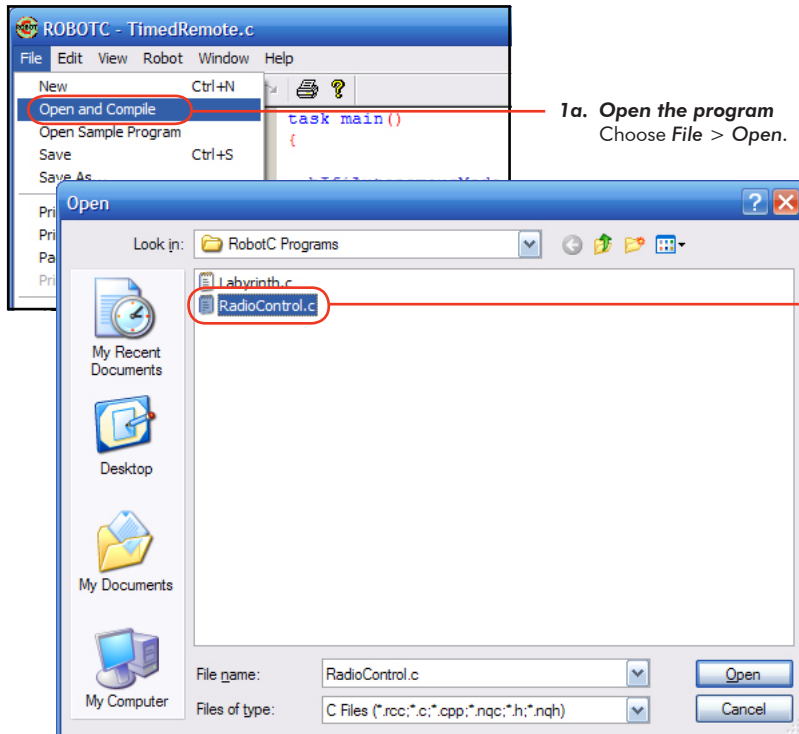
Since the default `time1[TimerName]` command reads in milliseconds, this poses a problem for us. Two minutes is 120 seconds, or 120,000 milliseconds. 120,000 is greater than 32767 and therefore cannot be expressed using the `time1` command. The `time10` command must be used instead.

# Radio

## Timers Using Timers

In this lesson, you will set up and use a Timer to limit the amount of time your robot remains in Radio Control mode, as required by the game rules.

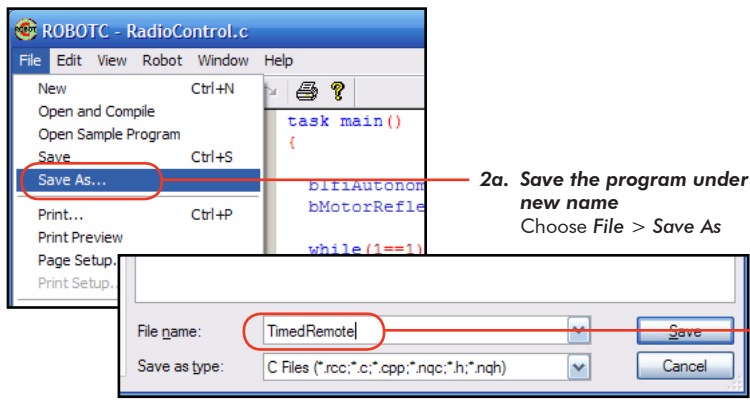
1. Open the RadioControl program.



**1a. Open the program**  
Choose *File* > *Open*.

**1b. Select the RadioControl program**  
Find *RadioControl* in the directory where you normally save your programs and double-click to open it (or select it and click *Open*).

2. Save this program under a new name so we can make modifications to it without disturbing the original.



**2a. Save the program under a new name**  
Choose *File* > *Save As*

**2b. Save As "TimedRemote"**  
Save your program in the usual folder, with the name *TimedRemote*.

## Radio

### Timers Using Timers *(cont.)*

#### Checkpoint

The current program allows you to control your robot using the remote control as long as the robot remains powered on. In order to be tournament legal, however, the robot must stop responding to commands no more than *two minutes* after the starting signal.

In the program as written, the robot continues updating motor powers forever because the **while()** loop repeats the **motor** commands over and over *forever*. If the loop were to only repeat those commands for a *certain amount of time*, then motor control would cease *after* the time had elapsed (and the **motor** commands were no longer being run by the loop).

```
while(1 == 1)
```

While loops will repeat *as long as their (condition)s remain true*. While (1 == 1) literally means “while 1 is equal to 1 is true”. Of course 1 is always equal to 1! Whatever is being controlled by this statement will loop forever. Therefore, the loop repeats forever. To make the loop only repeat for a certain amount of time, we need to change the (condition) to only be *true while the elapsed time is less than the desired time*.

#### Helpful Hints

For additional information and review, see the reference pages for Boolean Logic and **while** Loops.

```
while(time10[T1] < 500)
```

This **while()** loop will continue looping *while* (as long as) the time value in Timer T1 remains less than 500 hundredths of a second, or 5 seconds.

Let’s start by implementing a short-duration test program designed to disable radio control after 5 seconds (using the (condition) shown above). Once we have the program working with a 5-second cutoff, we’ll change it up to the tournament regulation 2 minutes.

## Radio

### Timers Using Timers (cont.)

3. Timers should always be reset before use. They begin counting immediately after they are reset. Add the `ClearTimer()` command just before the robot enters the Radio Control `while()` loop.

```

1 task main()
2 {
3
4     bIfiAutonomousMode = false;
5     bMotorReflected[port2] = 1;
6
7     ClearTimer(T1);
8     while(1 == 1)
9     {
10        motor[port3] = vexRT[Ch3];
11        motor[port2] = vexRT[Ch2];
12    }
13 }
```

**3. Add this code**

Clear the timer T1 so that it starts counting from the beginning of the radio control period.

4. Change the (condition) of the `while()` loop to check the Timer. The `while()` loop should run while the Timer value is still below (less than, `<`) 5 seconds.

```

1 task main()
2 {
3
4     bIfiAutonomousMode = false;
5     bMotorReflected[port2] = 1;
6
7     ClearTimer(T1);
8     while(time10[T1] < 500)
9     {
10        motor[port3] = vexRT[Ch3];
11        motor[port2] = vexRT[Ch2];
12    }
13
14    motor[port3] = 0;
15    motor[port2] = 0;
16 }
```

**4a. Modify this code**

Replace the "infinite" (`1==1`) condition with the more appropriate "while" condition (`time10[T1] < 500`). This allows the `while()` loop to continue repeating as long as the accumulated time in T1 is less than 500 hundredths of a second (5 seconds).

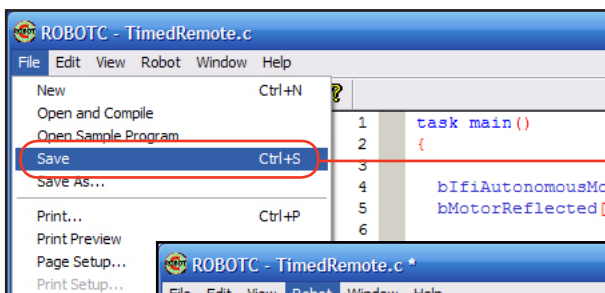
**4b. Add this code**

Manually ensure that the robot comes to a stop after the loop ends.

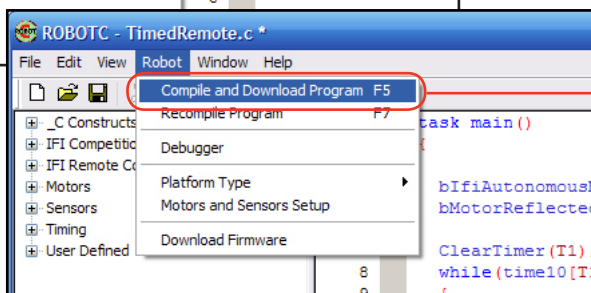
## Radio

### Timers Using Timers (cont.)

5. Save and download your program.



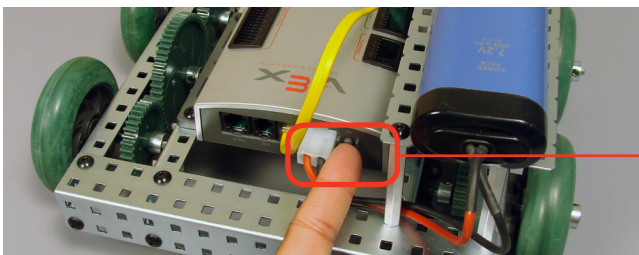
**5a. Save your program**  
Select File > Save.



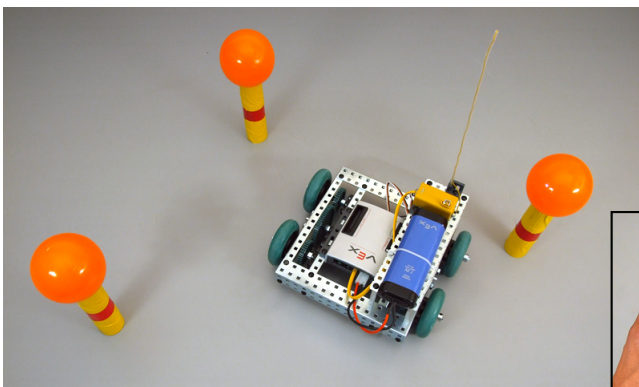
**5b. Compile and Download**

Make sure your robot is turned on and plugged in with the USB cable. Choose Robot > Compile and Download to download your program to the robot.

6. Run and test the program.



**6a. Run the program**  
Turn the robot off, then back on.



**6b. Drive the robot**

Drive around using the Transmitter, and use a stopwatch to see how long you remain in control of the robot. What happened after five seconds?



### Checkpoint

Your Transmitter should work to control the robot for exactly 5 seconds, and then stop.

## Radio

## Timers Using Timers (cont.)

7. Finally, change your program to use the actual two-minute time limit instead.

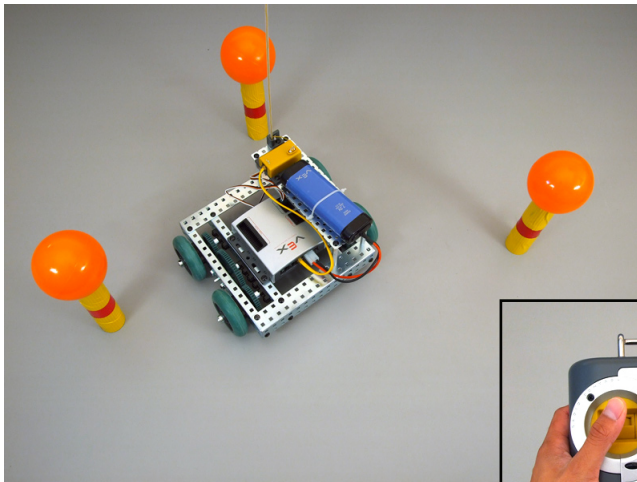
```

1 task main()
2 {
3
4     bIfiAutonomousMode = false;
5     bMotorReflected[port2] = 1;
6
7     ClearTimer(T1);
8     while(time10[T1] < 12000)
9     {
10        motor[port3] = vexRT[Ch3];
11        motor[port2] = vexRT[Ch2];
12    }
13
14    motor[port3] = 0;
15    motor[port2] = 0;
16 }

```

**7a. Modify this code**

Change the 500-hundredths (5 second) cutoff to 12,000 hundredths of a second (2 minutes).

**7b. Download, run and test**

The robot should now be controllable for two minutes starting at the beginning of the program. It should then automatically disable operator control.





## Radio

### Timers Using Timers (cont.)

8. There is still a place for the `wait1Msec` command in our program, at least for now. Use the `wait1Msec` command to insert a 2-second pause for the human operator to move his or her hand clear of the robot at the beginning of the program, just as we did in the Labyrinth.

```
1 task main()
2 {
3     wait1Msec(2000);
4     bIfiAutonomousMode = false;
5     bMotorReflected[port2] = 1;
6
7     ClearTimer(T1);
8     while(time10[T1] < 12000)
9     {
10        motor[port3] = vexRT[Ch3];
11        motor[port2] = vexRT[Ch2];
12    }
13
14    motor[port3] = 0;
15    motor[port2] = 0;
16 }
```

**8. Add this code**

Add a 2-second pause at the beginning of the program. This gives the human operator time to move out of the robot's way after turning it on.

This also helps to avoid the touching penalty for accidentally having a hand on the robot after the start-of-round signal.

#### End of Section

Your robot is now compliant with the challenge time limit rules! The next step is to build and program controls for an actuator to allow your robot to pick up the mines.