volatile

[Variable Scope & Qualifiers]

Description

volatile is a keyword known as a variable *qualifier*, it is usually used before the datatype of a variable, to modify the way in which the compiler and subsequent program treat the variable.

Declaring a variable volatile is a directive to the compiler. The compiler is software which translates your C/C++ code into the machine code, which are the real instructions for the Atmega chip in the Arduino.

Specifically, it directs the compiler to load the variable from RAM and not from a storage register, which is a temporary memory location where program variables are stored and manipulated. Under certain conditions, the value for a variable stored in registers can be inaccurate.

A variable should be declared volatile whenever its value can be changed by something beyond the control of the code section in which it appears, such as a concurrently executing thread. In the Arduino, the only place that this is likely to occur is in sections of code associated with interrupts, called an interrupt service routine.

int or long volatiles

If the volatile variable is bigger than a byte (e.g. a 16 bit int or a 32 bit long), then the microcontroller can not read it in one step, because it is an 8 bit microcontroller. This means that while your main code section (e.g. your loop) reads the first 8 bits of the variable, the interrupt might already change the second 8 bits. This will produce random values for the variable.

Remedy:

While the variable is read, interrupts need to be disabled, so they can’t mess with the bits, while they are read. There are several ways to do this:

1. LANGUAGE [noInterrupts](https://www.arduino.cc/reference/en/language/functions/interrupts/nointerrupts)
2. use the ATOMIC\_BLOCK macro. Atomic operations are single MCU operations - the smallest possible unit.

Example Code

// toggles LED when interrupt pin changes state

int pin = 13;

volatile byte state = LOW;

void setup() {

pinMode(pin, OUTPUT);

attachInterrupt(digitalPinToInterrupt(2), blink, CHANGE);

}

void loop() {

digitalWrite(pin, state);

}

void blink() {

state = !state;

}

#include <util/atomic.h> // this library includes the ATOMIC\_BLOCK macro.

volatile int input\_from\_interrupt;

ATOMIC\_BLOCK(ATOMIC\_RESTORESTATE) {

// code with interrupts blocked (consecutive atomic operations will not get interrupted)

int result = input\_from\_interrupt;

}