TinyStepper_28BYJ_48:

This Arduino library is used to control one or more *28BYJ-48* stepper motors. These motors are very small with a built in gear reduction. The advantage of these motors is that they are inexpensive and easily available on Amazon. Their downside is that they're very slow (0.3 revolutions/second max), and only suitable for light-duty applications.

The primary advantage of using this stepper library with the 28BYJ-48 motors is easy control and faster speeds. This is achieved by accelerating and decelerating the motor as they travel to their destination.

The 28BYJ-48 stepper motor with gear head has 2048 steps/revolution.

Absolute vs relative moves:

The functions that make a motor move come in two flavors: *Absolute* and *Relative*. Relative moves will use a coordinate system that is relative to the motor's current position. For example moving relative 200 steps, then another 200, then another 200, will turn 600 steps in total (and end up at an absolute position of 600).

Absolute moves use a coordinate system that is referenced to the original position of the motor when it is first turned on. First issuing an absolute move to position 200 will rotate forward one rotation. Then running the next absolute move to position 400 will turn just one more revolution in the same direction. Finally, an absolute move to position 0 will rotate backward two revolutions, back to the starting point.

Blocking vs Non-blocking function calls:

The easiest way to tell a stepper motor to move is using a function like this: moveToPositionInSteps(2048)

This function will rotate the motor until it gets to coordinate 2048, but the function call doesn't return until the motion is complete. This is a *Blocking* function.

Blocking functions are a limitation if you want to do other things while the motor is running. Examples of "other things" might be to: 1) Run two motors at once. 2) Check if a button is press, then decelerate to a stop. 3) Turn on a solenoid when the motor moves past position 850. 4) Flash a strobe every 1024 steps. 5) Run continuously as long as a temperature is below 125 degrees. To do these types of things *Non-blocking* functions are used. Here is an example:

```
//
// setup the motor so that it will rotate 10 revolutions, note: this
// command does not start moving yet
//
stepper.setupMoveInSteps(2048 * 10);

//
// now execute the move, looping until the motor has finished
//
```

```
  while(!stepper.motionComplete())
  {
    stepper.processMovement();       // this call moves themotor

    //
    // check if motor has moved past position 4096, if so turn On the LED
    //
    if (stepper.getCurrentPositionInSteps() == 4096)
      digitalWrite(LED_PIN, HIGH);
  }
```

There are some limitations to consider:

1. The code that you run while the stepper is moving needs to execute VERY fast. Perhaps no longer than 0.05 milliseconds.
2. The only change that you can make to a motion once it starts moving is to decelerate to a stop using *setupStop()*. Note: This library does not allow changing the target position or speed while the motor is moving.

Usage examples:

```
Near the top of the program, add:
    include "TinyStepper_28BYJ_48.h"

For each stepper, declare a global object outside of all functions as follows:
    TinyStepper_28BYJ_48 stepper1
    TinyStepper_28BYJ_48 stepper2;

In Setup(), assign IO pins used for Step and Direction:
    stepper1.connectToPins(2, 3, 4, 5);
    stepper2.connectToPins(6, 7, 8, 9);
```

```
Move one motor in units of steps:
    //
    // set the speed in steps/second and acceleration in steps/second/second
    //
    stepper1.setSpeedInStepsPerSecond(256);
    stepper1.setAccelerationInStepsPerSecondPerSecond(512);

    //
    // move 2048 steps (one revolution) in the backward direction
    //
    stepper1.moveRelativeInSteps(-2048);

    //
    // move to an absolute position of 2048 steps
    //
    stepper1.moveToPositionInSteps(2048);
```

```
Move two motors in units of steps:
    //
    // set the speed in rotations/second and acceleration in
    // steps/second/second
    //
    stepper1.setSpeedInStepsPerSecond(300);
    stepper1.setAccelerationInStepsPerSecondPerSecond(1000);
    stepper2.setSpeedInStepsPerSecond(300);
    stepper2.setAccelerationInStepsPerSecondPerSecond(1000);

    //
    // setup motor 1 to move backward 1.5 revolutions, this step does not
    // actually move the motor
    //
    stepper1.setupRelativeMoveInSteps(2048 * -1.5);

    //
    // setup motor 2 to move forward 3 revolutions, this step does not
    // actually move the motor
    //
    stepper2.setupRelativeMoveInRevolutions(2048 * 3.0);

    //
    // execute the moves
    //
    while((!stepper1.motionComplete()) || (!stepper2.motionComplete()))
    {
      stepper1.processMovement();
      stepper2.processMovement();
    }
```

The library of functions

```
//
// connect the stepper object to the IO pins
//  Enter:  in1PinNumber = IO pin number for motor wire 1 (blue)
//          in2PinNumber = IO pin number for motor wire 2 (pink)
//          in3PinNumber = IO pin number for motor wire 3 (yellow)
//          in4PinNumber = IO pin number for motor wire 4 (orange)
//
void connectToPins(byte in1PinNumber, byte in2PinNumber, byte in3PinNumber,
                   byte in4PinNumber)


//
// set the current position of the motor in steps, this does not move the motor
// Note: This function should only be called when the motor is stopped
```

```
//    Enter:  currentPositionInSteps = the new position of the motor in steps
//
void setCurrentPositionInSteps(long currentPositionInSteps)


//
// get the current position of the motor in steps, this functions is updated
// while the motor moves
//  Exit:  a signed motor position in steps returned
//
long getCurrentPositionInSteps()


//
// setup a "Stop" to begin the process of decelerating from the current velocity to
// zero, decelerating requires calls to processMove() until the move is complete
//
void setupStop()


//
// set the maximum speed in steps/second, this is the maximum speed reached
// while accelerating
// Note: this can only be called when the motor is stopped
//  Enter:  speedInStepsPerSecond = speed to accelerate up to, units in steps/second
//
void setSpeedInStepsPerSecond(float speedInStepsPerSecond)


//
// set the rate of acceleration in steps/second/second
// Note: this can only be called when the motor is stopped
//  Enter:  accelerationInStepsPerSecondPerSecond = rate of acceleration, units in
//          steps/second/second
//
void setAccelerationInStepsPerSecondPerSecond(
                          float accelerationInStepsPerSecondPerSecond)


//
// move relative to the current position in steps, this function does not return
// until the move is complete
//  Enter:  distanceToMoveInSteps = signed distance to move relative to the current
//          position in steps
//
void moveRelativeInSteps(long distanceToMoveInSteps)


//
// setup a move relative to the current position, units are in steps, no motion
// occurs until processMove() is called
// Note: this can only be called when the motor is stopped
//  Enter:  distanceToMoveInSteps = signed distance to move relative to the current
```

```
//          position in steps
//
void setupRelativeMoveInSteps(long distanceToMoveInSteps)


//
// move to the given absolute position in steps, this function does not return until
// the move is complete
//  Enter:  absolutePositionToMoveToInSteps = signed absolute position to move to in
//          units of steps
//
void moveToPositionInSteps(long absolutePositionToMoveToInSteps)


//
// setup a move with units are in steps, no motion occurs until processMove() is called
// Note: this can only be called when the motor is stopped
//  Enter:  absolutePositionToMoveToInSteps = signed absolute position to move to in
//          units of steps
//
void setupMoveInSteps(long absolutePositionToMoveToInSteps)


//
// if it is time, move one step
//  Exit:  true returned if movement complete, false returned not a final target
//           position yet
//
bool processMovement(void)


//
// Get the current velocity of the motor in steps/second.  This functions is updated
// while it accelerates up and down in speed.  This is not the desired speed, but
// the speed the motor should be moving at the time the function is called.  This
// is a signed value and is negative when the motor is moving backwards.
// Note: This speed will be incorrect if the desired velocity is set faster than
// this library can generate steps, or if the load on the motor is too great for
// the amount of torque that it can generate.
//  Exit:  velocity speed in steps per second returned, signed
//
float getCurrentVelocityInStepsPerSecond()


//
// check if the motor has competed its move to the target position
//  Exit:  true returned if the stepper is at the target position
//
bool motionComplete()


//
// disable the motor, all the drive coils are turned off to save power
```

```
// and reduce heat when motor is not in motion, any movement command will
// automatically renable the stepper
//
void disableMotor()
```