

WPI Robovation Kit Framework

The WPI Framework (WPILib) is designed to make the job of programming the Innovation First Robovation kits easier than using the default code alone. The WPILib provides a number of functions that you can call from your C program that handle most of the bookkeeping of driving motors in simple robot designs and handling inputs and outputs.

What it does for you

The library solves several problems:

- The default IFI code requires that you have a main loop where Getdata is called at the top of the loop and Putdata at the bottom, and it has to be done every 16ms or the controller will report a fault and stop.
- The motor (and other values) range from 0-255 with 127 in the center. Most algorithms you write have zero as center or stopped and a negative value for backwards and a positive value for forward.
- Many of the hardware features are less than obvious for a beginning programmer such as getting/setting values from/to I/O ports.

WPILib normalizes all values inside of this -127-127 range for you as well as the bookkeeping associated with having motors running in opposite directions on opposite sides of the robot. In addition there is a slight modification of motor values to account for internal friction of the motors at low speed.

WPI also removes the requirement to call Getdata and Putdata in a loop to drive your robot. Getdata and Putdata are still called, but behind the scenes under control of a timer interrupt. Motor data is automatically sent to the master processor for you and inputs from the PWM Inputs are automatically retrieved without you having to do any work. WPILib provides a number of convenience functions to make operation of the robot a little easier for the novice programmer.

Initialization

Before using any of the WPILib routines you need to initialize the library. This operation sets up the library to handle processing of I/O data for you. This **must** be the first executable line of your main program. It may be in a function, but it has to be executed before any other use of the robot controller happens. Failure to call this function will cause almost everything else to fail.

```
void main(void)
{
    WPIInitialize();
    //
    // the rest of your program logic goes here
    // .
    // .
    // .
}
```

Motors

The WPILib automatically handles the bookkeeping of driving motors for you. Consider a typical robot with two motors on opposite sides of the robot. The motors have to be driven in opposite directions since they are typically mounted opposing each other. The actual motor values range from 0 to 255 with 127 being the center or OFF position. WPILib allows you to use a more convenient scale: -127 for full backwards, 0 for off, and 127 for full forwards.

Setting up for Driving

Before using any of the high level drive routines you should indicate which PWM Outputs on the robot controller are connected to the left motor(s) and which are connected to the right motor(s). This is done using a call to either `TwoWheelDrive(left, right)` or `FourWheelDrive(left, frontLeft, right, frontRight)`. These functions tell the library where the motors are connected to make the Drive functions work correctly. In a four wheel drive robot (one with 4 motors), both of the left motors will drive at the same speed and both of the right motors will drive at the same speed.

To control a 2 wheel drive robot where the left motor is connected to PWM Out 1 and the right motor is connected to PWMOut 2 use the following code:

```
void main(void)
{
    WPIInitialize();
    TwoWheelDrive(1, 2);
}
```

Driving Motors

Rather than having to control both motors independently the WPILib has functions to drive the robot given a speed and a direction. The `Drive(speed, direction)` will drive both motors at the same time. The speed values range from -127 to 127 for full reverse to full forward. The directions may be -127 to 127 depending on how fast a turn is required. What's actually happening is quite simple: the direction value is added to the speed for one wheel and subtracted from the speed for the other wheel. If the values are out of range, they are brought back into range at the maximum value allowed (127 or -127). Here are some examples:

Speed	Direction	Result
127	0	Full speed forward
-127	0	Full speed reverse
0	60	Turn on center of rotation with each wheel running at 60
60	60	Turn while moving forward (one wheel runs at 0, the other at 120)

You may also drive motors independently supplying values for the left and right motors. This is sometimes convenient for cases like tank steering where two inputs will be independently drive the two motors. In this case the `Motors(leftValue, rightValue)` function may be called. It will drive the left motors with `leftValue` and the right motors with `rightValue`.

You may also control a single PWM Output by calling `Motor(pwmPort, speed)` where you explicitly give a port number and speed to drive the motor. The speed is in the range -127 to 127.

PWM Inputs and Radio Control

Using the robots with the radio control is easy using the `PWMIn(port)` function. The value of the PWM Input connected to `port` is returned as a value from -127 to 127. This makes it especially convenient to write a simple program to drive the robot using single joystick drive. Assuming you connect the two drive motors to PWM Outputs 1 and 2, and the receiver ports 1 and 2 to the PWM Inputs 1 and 2 the following program will drive the robot under remote control:

```
void main(void)
{
    WPIInitialize();
    TwoWheelDrive(1, 2);
    while (1)
    {
        Drive(PWMIn(2), PWMIn(1));
    }
}
```

The line:

```
Drive(PWMIn(2), PWMIn(1));
```

Will take the input from the y axis on the right joystick (`PWMIn(2)`) and use it to set the driving speed and the x axis on the right joystick (`PWMIn(1)`) will control the turn rate. This makes for a simple program to drive using a single joystick.

Controlling Servos

There are two functions for controlling hobby servos such as the ones supplied with the R/C sets. The servos are connected to the PWM Outputs on the robot controller. You must designate which ports servos are connected to:

```
SetPortAsServo(port-number);
```

This function should be called once for every port that has a servo connected to it. To control the servo use the following function:

```
Servo(port-number, value);
```

The values supplied have a range of 0-255 and will set the servo shaft to a repeatable angle depending on the value. The servos can operate through about 90 degrees of rotation.

Field Control – Enabled and Autonomous

Each match is 2:00 minutes long where the first 15 seconds is autonomous robot operation and the last 1:45 is under human control. The PWM Input values (from the r/c transmitter) will not be passed to the program during the autonomous period. Before and after the 2:00 minute match the robots will be disabled.

To facilitate this operation a hardware radio receiver will be provided just before the match start. This is not the same receiver as you use to drive the robot; it is separate and is used in addition to support field operation. It must be connected to digital/analog ports 13 and 14 so you should not use those ports for anything else.

The WPILib framework has been extended to include functions to allow the program to determine whether it is in autonomous/operator mode and if the robot is enabled/disabled. To determine if the robot is currently disabled or enabled use this function:

```
value = IsEnabled();  
or  
if (IsEnabled())...
```

The function will return a non-zero (true) value if the robot is enabled. To determine if you are in autonomous mode use this function:

```
value = IsAutonomous();  
or  
if (IsAutonomous())...
```

This function will return a non-zero (true) value if the robot is in autonomous mode. The motor functions described below will not operate if the robot is disabled. To get the robot to operate in operator mode (non-autonomous) for practice, you must put a jumper across the two outside pins of I/O Port 13. This will cause the robot to pass PWM Input values from the R/C Receiver to the software. **If you don't do this, the joysticks will not function!**

To determine how much time is left in autonomous mode there are two functions available to set and check the time. When the robot is first enabled you should reset the clock with the following function:

```
ResetClock();
```

This will set the match time clock on the robot to zero. If you don't do this, it will be the time since the robot was turned on. To read the clock values while the match is running you can use the function:

```
long time = SecondClock();
```

The SecondClock() function will return the number of seconds since the robot was turned on or since the last call to ResetClock(). You could use the time to control what operations you do in autonomous mode.

Utility Functions

There are several functions available that don't fit any of the previous categories that may be of use to you.

DebugPrintf(format-string, args...)

The DebugPrintf function has identical arguments and operation to printf except it will only print once every 100 milliseconds. This is to keep the console window from streaming repetitive information too fast. The function will check if it has been called in the last 100 ms and if it has, it will just return without printing anything.

You should be aware that if you have multiple DebugPrintf statements in your program, some of them might not print if a previous one printed recently. Only is DebugPrintf for repetitive printing of the same information.

SetDirection(port-number, direction);

This function will set one of the digital ports as an INPUT or OUTPUT port before you use it. Use the values INPUT or OUTPUT for the direction of the port. Be aware the the WPILib sets the first 8 ports to analog (1-8) by default and the remainder are set to digital INPUT ports.

SetPort(port-number, value);

int value = GetPort(port-number);

These functions will read and write digital values from this port-number. The values returned will be 8 bit values.