Appendix C: WPILib Functions Reference

Robot Initialization

```
void IO_Initialization(void);
void Set_Number_of_Analog_Channels(unsigned char numberOfChannels);
unsigned char IsAutonomous(void);
unsigned char IsEnabled(void);
```

Vex

```
void DefineControllerIO(unsigned char numberOfAnalogChannels,
                        unsigned char p1, unsigned char p2, unsigned char p3,
                        unsigned char p4, unsigned char p5, unsigned char p6,
                        unsigned char p7, unsigned char p8, unsigned char p9,
                        unsigned char p10, unsigned char p11, unsigned char p12,
                        unsigned char p13, unsigned char p14, unsigned char p15,
                        unsigned char p16);
void SetCompetitionMode(unsigned char autonomousTime,
                        unsigned char operatorTime);
```

FRC

```
void DefineControllerIO(unsigned char p1, unsigned char p2,
      unsigned char p3, unsigned char p4, unsigned char p5,
unsigned char p6, unsigned char p7, unsigned char p8,
      unsigned char p9, unsigned char p10, unsigned char p11,
      unsigned char p12, unsigned char p13, unsigned char p14,
      unsigned char p15, unsigned char p16, unsigned char p17,
      unsigned char p18);

void SetCompetitionMode(unsigned char mode);
```

General Purpose I/O

```
// direction values for SetDirection()
#define INPUT   1
#define OUTPUT  0

void SetDigitalOutput(unsigned char port, unsigned char value);

unsigned char GetDigitalInput(unsigned char port);

// set port direction (INPUT or OUTPUT)
void SetDirection(unsigned char port, unsigned char direction);
unsigned int GetAnalogInput(unsigned char port);
unsigned int Get_Analog_Value (unsigned char ADC_channel);
```

Debugging/Printing Output

```
// Non-graphical printing
void PrintToScreen(rom const char *fmt, ...);

// Graphical printing (using easyC terminal window in graphic mode only)
void SetGDWaitTime(unsigned time);
void ResetGD(void);
void ClearGD(unsigned char ucRow1, unsigned char ucCol1, unsigned char ucRow2,
             unsigned char ucCol2, unsigned char ucFrame);
```

```
void PrintTextToGD(unsigned char ucRow, unsigned char ucCol, unsigned long
                   ulColor, rom const char *szText, ...);
void PrintFrameToGD(unsigned char ucRow1, unsigned char ucCol1, unsigned char
                    ucRow2, unsigned char ucCol2, unsigned long ulColor);
```

Encoders

```
// Single line encoders (Vex encoders or other pulse counting applications)
void StartEncoder(unsigned char channel);
void StopEncoder(unsigned char channel);
long GetEncoder(unsigned char channel);
void PresetEncoder(unsigned char channel, long presetValue);

// Quadrature Encoder functions
void StartQuadEncoder(unsigned char channelA, unsigned char channelB,
                      unsigned char invert);
void StopQuadEncoder(unsigned char channelA, unsigned char channelB);
long GetQuadEncoder(unsigned char channelA, unsigned char channelB);
void PresetQuadEncoder(unsigned char channelA, unsigned char channelB,
                       long presetValue);

// Gear tooth sensor functions
void StartGTSensor(unsigned char port, unsigned char invert);
void StopGTSensor(unsigned char port);
long GetGTSensor(unsigned char port);
void PresetGTSensor(unsigned char port, long presetValue);
```

Interrupt Handling

```
// which edge will cause the ISR to be called
// int 1 & 2 done in hardware, 3-6 in software
#define RISING_EDGE 1
#define FALLING_EDGE 0

// Enabling and disabling interrupts
#define DISABLE_INTERRUPTS INTCONbits.GIEL = 0
#define ENABLE_INTERRUPTS INTCONbits.GIEL = 1

// Interrupt watchers latch state when interrupt occurs
void StartInterruptWatcher(unsigned char port, unsigned char edgeType);
void StopInterruptWatcher(unsigned char port);
unsigned char GetInterruptWatcher(unsigned char port);

// Registering interrupt handler function (ISR)
void RegisterInterruptHandler(unsigned char port, unsigned char edgeType,
                      void (*handler)(unsigned char port, unsigned char
value));
void UnRegisterInterruptHandler(unsigned char port);
void SetInterruptEdge(unsigned char port, unsigned char edgeType);
```

Ultrasonic Rangefinders

```
unsigned int GetUltrasonic(unsigned char echo, unsigned char ping);
void StartUltrasonic(unsigned char echo, unsigned char ping);
void StopUltrasonic(unsigned char echo, unsigned char ping);
```

Gyros

```
// these are gyro types that will be used for SetGyroType
#define ADXRS300    50
#define ADXRS150    125
#define ADXRS80     125

void InitGyro(unsigned char port);
void StartGyro(unsigned char port);
void StopGyro(unsigned char port);
int GetGyroAngle(unsigned char port);
void SetGyroType(unsigned char port, unsigned type);
void SetGyroDeadband(unsigned char port, char deadband);
```

Accelerometers

```
void InitAccelerometer(unsigned char port);
void StartAccelerometer(unsigned char port);
int GetAcceleration(unsigned char port);
void StopAccelerometer(unsigned char port);
```

Timing

```
// Suspend program while motors and interrupt handlers continue to run
void Wait(unsigned long ms);

// functions to support 6 timers
void StartTimer(unsigned char timerNumber);
void StopTimer(unsigned char timerNumber);
void PresetTimer(unsigned char timerNumber, unsigned long value);
unsigned long GetTimer(unsigned char timerNumber);

// Getting the time from the clock
unsigned GetSecondClock(void);   // time in seconds
unsigned long GetMsClock(void);  // time in milliseconds
unsigned GetGameTime(void);             // time since autonomous started
unsigned GetSecondClock(void);   // time in seconds
unsigned long GetUsClock(void);  // time in microseconds

// Timer interrupt handlers - called with interrupts disabled

// Timer interrupt handler function type
typedef void (*TimerHandler)(void);

// registering timer interrupt handler
void RegisterSingleTimer(unsigned long time, void (*handler)(void));
void RegisterRepeatingTimer(unsigned long time, void (*handler)(void));
void CancelTimer(void (*handler)(void));
```

Operator Interface

Operator Interface Status Displays (FRC only)

```
// OI Status LEDs definitions
#define PWM1_GREEN 0
#define PWM1_RED 1
#define PWM2_GREEN 2
#define PWM2_RED 3
#define RELAY1_GREEN 4
#define RELAY1_RED 5
#define RELAY2_GREEN 6
#define RELAY2_RED 7
#define SWITCH1_LED 8
#define SWITCH2_LED 9
#define SWITCH3_LED 10

// Operating status LEDs
void SetOILED(unsigned char led, unsigned char value);
void SetUserDisplay(unsigned char value);
```

Operator Interface Controls

Vex

```
Vex receiver to controller port numbers
#define PORT_1 1
#define PORT_2 2

// Vex transmitter channel numbers
#define CHANNEL_1 1
#define CHANNEL_2 2
#define CHANNEL_3 3
#define CHANNEL_4 4
#define CHANNEL_5 5
#define CHANNEL_6 6
#define CHANNEL_5_TOP 1
#define CHANNEL_5_BOTTOM 2
#define CHANNEL_6_TOP 3
#define CHANNEL_6_BOTTOM 4

// Vex functions for rear buttons (returns 1/0 for CHANNEL_5_TOP, etc.)
// these extra functions are for channels 5 and 6 controller back
// You can use GetRXInput instead and get back 0/255 for top/bottom or 127
// for compatibility with FRC
unsigned char GetOIDInput(unsigned char port, unsigned char channel);
unsigned char GetOIAInput(unsigned char port, unsigned char channel);

// Get a Vex receiver channel value
unsigned char GetRxInput(unsigned char port, unsigned char channel);

// Vex simplified OI to drive functions
// These functions send the receiver to the motors ONCE (must be in a loop)
void Arcade2(unsigned char port,
                unsigned char moveChannel, unsigned char rotateChannel,
                unsigned char leftPWM, unsigned char rightPWM,
```

```
                unsigned char leftInvert, unsigned char rightInvert);
void Arcade4(unsigned char port,
             unsigned char ucMoveChannel, unsigned char ucRotateChannel,
             unsigned char ucLeftfrontPWM, unsigned char ucRightfrontPWM,
             unsigned char ucLeftrearPWM, unsigned char ucRightrearPWM,
             unsigned char ucLeftfrontInvert, unsigned char ucRightfrontInvert,
             unsigned char ucLeftrearInvert, unsigned char ucRightrearInvert);
void Tank2(unsigned char port,
             unsigned char leftChannel, unsigned char rightChannel,
             unsigned char leftPWM, unsigned char rightPWM,
             unsigned char leftInvert, unsigned char rightInvert);
void Tank4(unsigned char port,
             unsigned char ucLeftChannel, unsigned char ucRightChannel,
             unsigned char ucLeftfrontPWM, unsigned char ucRightfrontPWM,
             unsigned char ucLeftrearPWM, unsigned char ucRightrearPWM,
             unsigned char ucLeftfrontInvert, unsigned char ucRightfrontInvert,
             unsigned char ucLeftrearInvert, unsigned char ucRightrearInvert);
```

FRC

```
// FRC OI Port numbers
#define PORT_1 1
#define PORT_2 2
#define PORT_3 3
#define PORT_4 4

// FRC joystick functions
#define X_AXIS 1
#define Y_AXIS 2
#define WHEEL_AXIS 3
#define AUX_AXIS 4
#define TRIGGER_SW 1
#define TOP_SW 2
#define AUX1_SW 3
#define AUX2_SW 4
#define ALT_TRIGGER 5
#define ALT_THUMB 6
#define ALT_AUX1_SW 7
#define ALT_AUX2_SW 8

// FRC Relay functions
void SetRelay(unsigned char port, char forward, char reverse);
void OIToRelay(unsigned char port, unsigned char function,
               unsigned char relayNumber, unsigned char direction);

// Set the user display value
void SetUserByte(unsigned char index, unsigned char value);

// FRC return raw value from OI for digital or analog port
unsigned char GetOIDInput(unsigned char port, unsigned char function);
unsigned char GetOIAInput(unsigned char port, unsigned char function);

// FRC Simple OI to Drive functions
// these functions send OI values to a motor ONCE (must be in a loop)
void Arcade2(unsigned char movePort, unsigned char moveChannel,
               unsigned char rotatePort, unsigned char rotateChannel,
```

```
                unsigned char leftPWM, unsigned char rightPWM,
                unsigned char leftInvert, unsigned char rightInvert);
void Arcade4(unsigned char ucMovePort, unsigned char ucMoveChannel,
             unsigned char ucRotatePort, unsigned char ucRotateChannel,
             unsigned char ucLeftfrontPWM, unsigned char ucRightfrontPWM,
             unsigned char ucLeftrearPWM, unsigned char ucRightrearPWM,
             unsigned char ucLeftfrontInvert, unsigned char ucRightfrontInvert,
             unsigned char ucLeftrearInvert, unsigned char ucRightrearInvert);
void Tank2(unsigned char leftPort, unsigned char leftChannel,
             unsigned char rightPort, unsigned char rightChannel,
             unsigned char leftPWM, unsigned char rightPWM,
             unsigned char leftInvert, unsigned char rightInvert);
void Tank4(unsigned char ucLeftPort, unsigned char ucLeftChannel,
             unsigned char ucRightPort, unsigned char ucRightChannel,
             unsigned char ucLeftfrontPWM, unsigned char ucRightfrontPWM,
             unsigned char ucLeftrearPWM, unsigned char ucRightrearPWM,
             unsigned char ucLeftfrontInvert, unsigned char ucRightfrontInvert,
             unsigned char ucLeftrearInvert, unsigned char ucRightrearInvert);

// OI values that are send directly to motors or digital outputs
void OIToDOutput(unsigned char port, unsigned char function, unsigned char
dport);
void OIToPWM(unsigned char port, unsigned char function, unsigned char pwm,
              unsigned char invert);
```

CMU Camera

```
// returned data packet from camera
typedef struct {
    unsigned char mx;
    unsigned char my;
    unsigned char x1;
    unsigned char y1;
    unsigned char x2;
    unsigned char y2;
    unsigned char regionSize;
    unsigned char confidence;
    unsigned char pan;
    unsigned char tilt;
    unsigned char length;
    unsigned char sequence;
} TPacket;

// data structure to initialize the camera parameters
typedef rom struct {
    unsigned char redMin, redMax, greenMin, greenMax, blueMin, blueMax;
    unsigned char yCrCb;
    unsigned char noiseFilter;
    unsigned char aecEnable;
    unsigned char aecLevel;
    unsigned char agcEnable;
    unsigned char agcLevel;
    unsigned char saturation;
    unsigned char blueGain;
    unsigned char redGain;
```

```
    unsigned char brightness;
    unsigned char panRangeFar;
    unsigned char panRangeNear;
    unsigned char panStep;
    unsigned char tiltRangeFar;
    unsigned char tiltRangeNear;
    unsigned char tiltStep;
} CameraInitializationData;

// initialize the camera to one of an array of CameraInitializeData elements
// you need to call one of these two functions before starting the camera

// InitCamera – designed for easyC
void InitCamera(unsigned char cameraInitIndex);

// InitializeCamera – takes a struct of camera data (designed for C
programmers)
void InitializeCamera(CameraInitializationData *c);

// camera operation – must be started before data packets can be used
void StopCamera(void);
void StartCamera(void);

// Get the current camera data values
// Use nulls for unneeded addresses
void CaptureTrackingData(
            unsigned char *centerX,
            unsigned char *centerY,
            unsigned char *x1,
            unsigned char *y1,
            unsigned char *x2,
            unsigned char *y2,
            unsigned char *regionSize,
            unsigned char *confidence,
            unsigned char *pan,
            unsigned char *tilt);

// Get the current TPacket (camera data packet)
TPacket *CopyTrackingData(void);

// sends a reset to the camera
unsigned char ResetCameraState(void);

// sends a single command to the camera like printf
// “%” characters are replaced with the numeric byte values supplied as
//     arguments
unsigned char CmdToCamera(const rom char *format, ...);

// operates the CAMERA servos – not the servos attached to PWM ports
void SetServoTracking(unsigned char panTracking, unsigned char tiltTracking);
void SetServoPosition(unsigned char servo, unsigned char position);

// Initialize the camera before starting it
unsigned char GetCameraStatus(void);
void SetCameraDebugMode(unsigned char mode);
```

Pneumatics Pressure Switch (FRC Only)

```
void InitPressureSwitch(unsigned char pressureSwitchPort, unsigned char
relayPort);
```

Serial Port

```
#define BAUD_4800 0x0081
#define BAUD_9600 0x0040
#define BAUD_14400 0x01AD
#define BAUD_19200 0x0181
#define BAUD_28800 0x0156
#define BAUD_38400 0x0140
#define BAUD_57600 0x012A
#define BAUD_115200 0x0115

unsigned char ReadSerialPortOne(void);
void WriteSerialPortOne(unsigned char);
unsigned char ReadSerialPortTwo(void);
void WriteSerialPortTwo(unsigned char);
unsigned char GetSerialPort2ByteCount(void);
unsigned char GetSerialPort1ByteCount(void);
void OpenSerialPortOne(unsigned baudRate);
void OpenSerialPortTwo(unsigned baudRate);
```

Compass

```
void InitializeCompass(unsigned char port);
unsigned GetCompassHeading(void);
```

Miscellaneous Functions

```
unsigned char GetPacketNumber(void);
```

FRC

```
unsigned GetMainBattery(void);
unsigned GetBackupBattery(void);
```

Vex

```
unsigned char ReceivingData(unsigned char port);
```

Motor and Servo Functions

```
// Set a single PWM value
void SetPWM(unsigned char port, unsigned char speed);

// Drive functions that are based on motor and direction values ranging from:
//   -127 - +127 for full backwards to full forwards
//   -127 - +127 for full right to full left turns (see Drive function)
void SetInvertedMotor(unsigned char port);
void Motor(unsigned char pwmPort, int speed);
void Motors(int leftSpeed, int rightSpeed);
void Drive(int speed, int direction);

// must call one of these functions prior to calling the above functions
void TwoWheelDrive(unsigned char _leftMotor, unsigned char _rightMotor);
void FourWheelDrive(unsigned char _leftMotor, unsigned char _frontLeftMotor,
                    unsigned char _rightMotor,
                    unsigned char _frontRightMotor);
```