Modbus communication C/C++ code for serial port without libmodbus-dev library.

Sure, if you prefer to implement Modbus RTU communication over a serial port without using the libmodbus library, you’ll need to handle the low-level details yourself, including serial port communication and Modbus protocol specifics (framing, CRC, etc.).

Below is a basic example of Modbus RTU communication using the POSIX API to handle serial communication. This example demonstrates reading and writing holding registers.

**Basic Requirements**

* A serial port for communication (e.g., /dev/ttyUSB0).
* Knowledge of the Modbus protocol for framing, addressing, and CRC calculation.

**Serial Communication Setup**

First, we'll set up the serial port for communication.

**Serial Port Initialization**

#include <iostream>

#include <fcntl.h>

#include <errno.h>

#include <termios.h>

#include <unistd.h>

int openSerialPort(const char \*device) {

int fd = open(device, O\_RDWR | O\_NOCTTY | O\_NDELAY);

if (fd == -1) {

std::cerr << "Unable to open serial port: " << strerror(errno) << std::endl;

return -1;

}

struct termios options;

tcgetattr(fd, &options);

cfsetispeed(&options, B9600);

cfsetospeed(&options, B9600);

options.c\_cflag |= (CLOCAL | CREAD);

options.c\_cflag &= ~PARENB;

options.c\_cflag &= ~CSTOPB;

options.c\_cflag &= ~CSIZE;

options.c\_cflag |= CS8;

tcsetattr(fd, TCSANOW, &options);

return fd;

}

C++

Copy code

**Modbus RTU CRC Calculation**

Modbus RTU uses a CRC-16 for error checking. Here's how to implement it.

uint16\_t modbusCRC(uint8\_t \*buffer, int length) {

uint16\_t crc = 0xFFFF;

for (int pos = 0; pos < length; pos++) {

crc ^= buffer[pos]; // XOR byte into least sig. byte of crc

for (int i = 8; i != 0; i--) { // Loop over each bit

if ((crc & 0x0001) != 0) { // If the LSB is set

crc >>= 1; // Shift right and XOR 0xA001

crc ^= 0xA001;

} else { // Else LSB is not set

crc >>= 1; // Just shift right

}

}

}

return crc;

}

C++

Copy code

**Modbus RTU Frame Construction and Communication**

**Function to Write a Single Register**

bool writeRegister(int fd, uint8\_t slaveAddress, uint16\_t registerAddress, uint16\_t value) {

uint8\_t frame[8];

frame[0] = slaveAddress; // Slave Address

frame[1] = 0x06; // Function Code (Write Single Register)

frame[2] = (registerAddress >> 8) & 0xFF; // Register Address High Byte

frame[3] = registerAddress & 0xFF; // Register Address Low Byte

frame[4] = (value >> 8) & 0xFF; // Value High Byte

frame[5] = value & 0xFF; // Value Low Byte

uint16\_t crc = modbusCRC(frame, 6);

frame[6] = crc & 0xFF; // CRC Low Byte

frame[7] = (crc >> 8) & 0xFF; // CRC High Byte

// Write frame to the serial port

if(write(fd, frame, 8) == -1) {

std::cerr << "Failed to write to the serial port: " << strerror(errno) << std::endl;

return false;

}

// Read response

uint8\_t response[8];

if (read(fd, response, 8) != 8) {

std::cerr << "Failed to read from the serial port: " << strerror(errno) << std::endl;

return false;

}

return true;

}

C++

Copy code

**Function to Read Holding Registers**

bool readRegisters(int fd, uint8\_t slaveAddress, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t \*buffer) {

uint8\_t frame[8];

frame[0] = slaveAddress; // Slave Address

frame[1] = 0x03; // Function Code (Read Holding Registers)

frame[2] = (

C++

Copy code

Copy

Good response

Bad response

continue

Certainly! Let's continue with the readRegisters function to read Modbus holding registers.

**Function to Read Holding Registers (Continued)**

frame[2] = (startAddress >> 8) & 0xFF; // Start Address High Byte

frame[3] = startAddress & 0xFF; // Start Address Low Byte

frame[4] = (numRegisters >> 8) & 0xFF; // Number of Registers High Byte

frame[5] = numRegisters & 0xFF; // Number of Registers Low Byte

uint16\_t crc = modbusCRC(frame, 6);

frame[6] = crc & 0xFF; // CRC Low Byte

frame[7] = (crc >> 8) & 0xFF; // CRC High Byte

// Write frame to the serial port

if(write(fd, frame, 8) == -1) {

std::cerr << "Failed to write to the serial port: " << strerror(errno) << std::endl;

return false;

}

// Calculate expected response length: Slave Address + Function Code + Byte Count + Data + CRC

int responseLength = 5 + 2 \* numRegisters;

uint8\_t response[responseLength];

// Read response from the serial port

if (read(fd, response, responseLength) != responseLength) {

std::cerr << "Failed to read from the serial port: " << strerror(errno) << std::endl;

return false;

}

// Verify CRC of the response

uint16\_t responseCRC = (response[responseLength - 1] << 8) | response[responseLength - 2];

if (modbusCRC(response, responseLength - 2) != responseCRC) {

std::cerr << "CRC check failed for received response" << std::endl;

return false;

}

// Extract the data

for (int i = 0; i < numRegisters; ++i) {

buffer[i] = (response[3 + 2 \* i] << 8) | response[4 + 2 \* i];

}

return true;

}

C++

Copy code

**Example Main Function**

Finally, let's bring everything together in the main function to demonstrate writing and reading registers via Modbus RTU over a serial port.

int main() {

// Open the serial port

const char\* device = "/dev/ttyUSB0";

int fd = openSerialPort(device);

if (fd == -1) {

return -1;

}

uint8\_t slaveAddress = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(fd, slaveAddress, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(fd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(fd, slaveAddress, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(fd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

}

// Close the serial port

close(fd);

return 0;

}

C++

Copy code

**Explanation**

1. **Serial Port Initialization**: The openSerialPort function configures the serial port for 9600 baud, no parity, 8 data bits, and 1 stop bit (9600-N-8-1).
2. **CRC Calculation**: The modbusCRC function calculates the Modbus RTU CRC-16 checksum.
3. **Write Register**: The writeRegister function constructs a Modbus request to write a single holding register and sends it to the serial port.
4. **Read Registers**: The readRegisters function constructs a Modbus request to read multiple holding registers and reads the response from the serial port.
5. **Main Function**: In the main function, the serial port is opened, a single register is written, multiple registers are read, and the results are printed.

**Notes**

* **Permission to Serial Port**: Ensure your user has the necessary permissions to access the serial port.

Copy

Good response

Bad response

continue

Certainly! Here are a few more considerations and suggestions to make the code more robust and functional:

**Handling Serial Port Errors and Timeout**

You might want to handle serial port timeouts and read/write errors more gracefully:

**Setting Serial Port Timeouts**

void setSerialPortTimeouts(int fd, int readTimeoutMs, int writeTimeoutMs) {

struct termios options;

tcgetattr(fd, &options);

// Read timeout

options.c\_cc[VMIN] = 0; // Minimum number of characters to read

options.c\_cc[VTIME] = readTimeoutMs / 100; // Timeout in deciseconds

// Apply the settings

tcsetattr(fd, TCSANOW, &options);

}

C++

Copy code

**Using Timeout in Communication**

Setting timeouts on serial communication can prevent the program from blocking indefinitely if the slave device does not respond.

**Improved Error Handling and Logging**

It's good practice to enhance error messages for clarity and log them if necessary.

**Complete Revamped Example**

Here is an enhanced version of the main Modbus communication program with improved error handling and timeouts.

#include <iostream>

#include <fcntl.h>

#include <errno.h>

#include <termios.h>

#include <unistd.h>

#include <cstring>

// Function prototypes

int openSerialPort(const char \*device);

void setSerialPortTimeouts(int fd, int readTimeoutMs, int writeTimeoutMs);

uint16\_t modbusCRC(uint8\_t \*buffer, int length);

bool writeRegister(int fd, uint8\_t slaveAddress, uint16\_t registerAddress, uint16\_t value);

bool readRegisters(int fd, uint8\_t slaveAddress, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t \*buffer);

int main() {

// Open the serial port

const char\* device = "/dev/ttyUSB0";

int fd = openSerialPort(device);

if (fd == -1) {

return -1;

}

// Set timeouts (optional, adjust as needed)

setSerialPortTimeouts(fd, 1000, 1000);

uint8\_t slaveAddress = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(fd, slaveAddress, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(fd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(fd, slaveAddress, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(fd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

}

// Close the serial port

close(fd);

return 0;

}

int openSerialPort(const char \*device) {

int fd = open(device, O\_RDWR | O\_NOCTTY | O\_NDELAY);

if (fd == -1) {

std::cerr << "Unable to open serial port: " << strerror(errno) << std::endl;

return -1;

}

struct termios options;

tcgetattr(fd, &options);

cfsetispeed(&options, B9600);

cfsetospeed(&options, B9600);

options.c\_cflag |= (CLOCAL | CREAD);

options.c\_cflag &= ~PARENB;

options.c\_cflag &= ~CSTOPB;

options.c\_cflag &= ~CSIZE;

options.c\_cflag |= CS8;

tcsetattr(fd, TCSANOW, &options);

return fd;

}

void setSerialPortTimeouts(int fd, int readTimeoutMs, int writeTimeoutMs) {

struct termios options;

tcgetattr(fd, &options);

// Read timeout

options.c\_cc[VMIN] = 0; // Minimum number of characters to read

options.c\_cc[VTIME] = readTimeoutMs / 100; // Timeout in deciseconds

// Apply the settings

tcsetattr(fd, TCSANOW, &options);

}

uint16\_t modbusCRC(uint8\_t \*buffer, int length) {

uint16\_t crc = 0xFFFF;

for (int pos = 0; pos <

C++

Copy code

Copy

Good response

Bad response

continue

Sure, here is the continuation of the modbusCRC function and the remaining code for reading and writing registers:

for (int pos = 0; pos < length; pos++) {

crc ^= buffer[pos]; // XOR byte into least sig. byte of crc

for (int i = 8; i != 0; i--) { // Loop over each bit

if ((crc & 0x0001) != 0) { // If the LSB is set

crc >>= 1; // Shift right and XOR 0xA001

crc ^= 0xA001;

} else { // Else LSB is not set

crc >>= 1; // Just shift right

}

}

}

return crc;

}

bool writeRegister(int fd, uint8\_t slaveAddress, uint16\_t registerAddress, uint16\_t value) {

uint8\_t frame[8];

frame[0] = slaveAddress; // Slave Address

frame[1] = 0x06; // Function Code (Write Single Register)

frame[2] = (registerAddress >> 8) & 0xFF; // Register Address High Byte

frame[3] = registerAddress & 0xFF; // Register Address Low Byte

frame[4] = (value >> 8) & 0xFF; // Value High Byte

frame[5] = value & 0xFF; // Value Low Byte

uint16\_t crc = modbusCRC(frame, 6);

frame[6] = crc & 0xFF; // CRC Low Byte

frame[7] = (crc >> 8) & 0xFF; // CRC High Byte

// Write frame to the serial port

if(write(fd, frame, 8) == -1) {

std::cerr << "Failed to write to the serial port: " << strerror(errno) << std::endl;

return false;

}

// Read response

uint8\_t response[8];

if (read(fd, response, 8) != 8) {

std::cerr << "Failed to read from the serial port: " << strerror(errno) << std::endl;

return false;

}

// Verify CRC of the response

uint16\_t responseCRC = (response[7] << 8) | response[6];

if (modbusCRC(response, 6) != responseCRC) {

std::cerr << "CRC check failed for received response" << std::endl;

return false;

}

return true;

}

bool readRegisters(int fd, uint8\_t slaveAddress, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t \*buffer) {

uint8\_t frame[8];

frame[0] = slaveAddress; // Slave Address

frame[1] = 0x03; // Function Code (Read Holding Registers)

frame[2] = (startAddress >> 8) & 0xFF; // Start Address High Byte

frame[3] = startAddress & 0xFF; // Start Address Low Byte

frame[4] = (numRegisters >> 8) & 0xFF; // Number of Registers High Byte

frame[5] = numRegisters & 0xFF; // Number of Registers Low Byte

uint16\_t crc = modbusCRC(frame, 6);

frame[6] = crc & 0xFF; // CRC Low Byte

frame[7] = (crc >> 8) & 0xFF; // CRC High Byte

// Write frame to the serial port

if(write(fd, frame, 8) == -1) {

std::cerr << "Failed to write to the serial port: " << strerror(errno) << std::endl;

return false;

}

// Calculate expected response length: Slave Address + Function Code + Byte Count + Data + CRC

int responseLength = 5 + 2 \* numRegisters;

uint8\_t response[responseLength];

// Read response from the serial port

if (read(fd, response, responseLength) != responseLength) {

std::cerr << "Failed to read from the serial port: " << strerror(errno) << std::endl;

return false;

}

// Verify CRC of the response

uint16\_t responseCRC = (response[responseLength - 1] << 8) | response[responseLength - 2];

if (modbusCRC(response, responseLength - 2) !=

C++

Copy code

Copy

Good response

Bad response

continue

Certainly! Let's continue where we left off by completing the readRegisters function and closing with a complete, runnable example.

**Completing the readRegisters Function**

Here is the continuation and completion of the readRegisters function:

if (modbusCRC(response, responseLength - 2) != responseCRC) {

std::cerr << "CRC check failed for received response" << std::endl;

return false;

}

// Extract the data

for (int i = 0; i < numRegisters; ++i) {

buffer[i] = (response[3 + 2 \* i] << 8) | response[4 + 2 \* i];

}

return true;

}

C++

Copy code

**Revised Complete Example**

Combining all parts together, here is the complete revised example code:

#include <iostream>

#include <fcntl.h>

#include <errno.h>

#include <termios.h>

#include <unistd.h>

#include <cstring>

// Function prototypes

int openSerialPort(const char \*device);

void setSerialPortTimeouts(int fd, int readTimeoutMs, int writeTimeoutMs);

uint16\_t modbusCRC(uint8\_t \*buffer, int length);

bool writeRegister(int fd, uint8\_t slaveAddress, uint16\_t registerAddress, uint16\_t value);

bool readRegisters(int fd, uint8\_t slaveAddress, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t \*buffer);

int main() {

// Open the serial port

const char\* device = "/dev/ttyUSB0";

int fd = openSerialPort(device);

if (fd == -1) {

return -1;

}

// Set timeouts (optional, adjust as needed)

setSerialPortTimeouts(fd, 1000, 1000);

uint8\_t slaveAddress = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(fd, slaveAddress, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(fd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(fd, slaveAddress, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(fd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

}

// Close the serial port

close(fd);

return 0;

}

int openSerialPort(const char \*device) {

int fd = open(device, O\_RDWR | O\_NOCTTY | O\_NDELAY);

if (fd == -1) {

std::cerr << "Unable to open serial port: " << strerror(errno) << std::endl;

return -1;

}

struct termios options;

tcgetattr(fd, &options);

cfsetispeed(&options, B9600);

cfsetospeed(&options, B9600);

options.c\_cflag |= (CLOCAL | CREAD);

options.c\_cflag &= ~PARENB;

options.c\_cflag &= ~CSTOPB;

options.c\_cflag &= ~CSIZE;

options.c\_cflag |= CS8;

tcsetattr(fd, TCSANOW, &options);

return fd;

}

void setSerialPortTimeouts(int fd, int readTimeoutMs, int writeTimeoutMs) {

struct termios options;

tcgetattr(fd, &options);

// Read timeout

options.c\_cc[VMIN] = 0; // Minimum number of characters to read

options.c\_cc[VTIME] = readTimeoutMs / 100; // Timeout in deciseconds

// Apply the settings

tcsetattr(fd, TCSANOW, &options);

}

uint16\_t modbusCRC(uint8\_t \*buffer, int length) {

uint16\_t crc = 0xFFFF;

for (int pos = 0; pos < length; pos++) {

crc ^= buffer[pos]; // XOR byte into least sig. byte of crc

for (int i = 8; i != 0; i--) { // Loop over each bit

if ((crc & 0x0001) != 0

C++

Copy code

Copy

Good response

Bad response

continue

Certainly! Here is the continuation of the modbusCRC function and the complete code all in one:

if ((crc & 0x0001) != 0) { // If the LSB is set

crc >>= 1; // Shift right and XOR 0xA001

crc ^= 0xA001;

} else { // Else LSB is not set

crc >>= 1; // Just shift right

}

}

}

return crc;

}

bool writeRegister(int fd, uint8\_t slaveAddress, uint16\_t registerAddress, uint16\_t value) {

uint8\_t frame[8];

frame[0] = slaveAddress; // Slave Address

frame[1] = 0x06; // Function Code (Write Single Register)

frame[2] = (registerAddress >> 8) & 0xFF; // Register Address High Byte

frame[3] = registerAddress & 0xFF; // Register Address Low Byte

frame[4] = (value >> 8) & 0xFF; // Value High Byte

frame[5] = value & 0xFF; // Value Low Byte

uint16\_t crc = modbusCRC(frame, 6);

frame[6] = crc & 0xFF; // CRC Low Byte

frame[7] = (crc >> 8) & 0xFF; // CRC High Byte

// Write frame to the serial port

if(write(fd, frame, 8) == -1) {

std::cerr << "Failed to write to the serial port: " << strerror(errno) << std::endl;

return false;

}

// Read response

uint8\_t response[8];

if (read(fd, response, 8) != 8) {

std::cerr << "Failed to read from the serial port: " << strerror(errno) << std::endl;

return false;

}

// Verify CRC of the response

uint16\_t responseCRC = (response[7] << 8) | response[6];

if (modbusCRC(response, 6) != responseCRC) {

std::cerr << "CRC check failed for received response" << std::endl;

return false;

}

return true;

}

bool readRegisters(int fd, uint8\_t slaveAddress, uint16\_t startAddress, uint16\_t numRegisters, uint16\_t \*buffer) {

uint8\_t frame[8];

frame[0] = slaveAddress; // Slave Address

frame[1] = 0x03; // Function Code (Read Holding Registers)

frame[2] = (startAddress >> 8) & 0xFF; // Start Address High Byte

frame[3] = startAddress & 0xFF; // Start Address Low Byte

frame[4] = (numRegisters >> 8) & 0xFF; // Number of Registers High Byte

frame[5] = numRegisters & 0xFF; // Number of Registers Low Byte

uint16\_t crc = modbusCRC(frame, 6);

frame[6] = crc & 0xFF; // CRC Low Byte

frame[7] = (crc >> 8) & 0xFF; // CRC High Byte

// Write frame to the serial port

if(write(fd, frame, 8) == -1) {

std::cerr << "Failed to write to the serial port: " << strerror(errno) << std::endl;

return false;

}

// Calculate expected response length: Slave Address + Function Code + Byte Count + Data + CRC

int responseLength = 5 + 2 \* numRegisters;

uint8\_t response[responseLength];

// Read response from the serial port

if (read(fd, response, responseLength) != responseLength) {

std::cerr << "Failed to read from the serial port: " << strerror(errno) << std::endl;

return false;

}

// Verify CRC of the response

uint16\_t responseCRC = (response[responseLength - 1] << 8) | response[responseLength - 2];

if (modbusCRC(response, responseLength - 2) != responseCRC) {

std::cerr << "CRC check failed for received response" << std::endl;

return false;

}

// Extract the data

for (int i = 0; i < numRegisters; ++i) {

buffer[i] = (response

C++

Copy code

Copy

Good response

Bad response

continue

Sure, here is the continuation and completion of the readRegisters function, followed by the fully integrated example:

buffer[i] = (response[3 + 2 \* i] << 8) | response[4 + 2 \* i];

}

return true;

}

// Complete example for interacting with a MODBUS slave

int main() {

// Open the serial port

const char\* device = "/dev/ttyUSB0";

int fd = openSerialPort(device);

if (fd == -1) {

return -1;

}

// Set timeouts (optional, adjust as needed)

setSerialPortTimeouts(fd, 1000, 1000);

uint8\_t slaveAddress = 1;

uint16\_t registerAddress = 0;

uint16\_t value = 1234;

// Write a single holding register

if (!writeRegister(fd, slaveAddress, registerAddress, value)) {

std::cerr << "Failed to write register" << std::endl;

close(fd);

return -1;

}

// Read holding registers

const int numRegisters = 5;

uint16\_t buffer[numRegisters] = {0};

if (!readRegisters(fd, slaveAddress, registerAddress, numRegisters, buffer)) {

std::cerr << "Failed to read registers" << std::endl;

close(fd);

return -1;

}

// Print the read registers

std::cout << "Read registers:" << std::endl;

for (int i = 0; i < numRegisters; ++i) {

std::cout << "Register " << i << ": " << buffer[i] << std::endl;

}

// Close the serial port

close(fd);

return 0;

}

int openSerialPort(const char \*device) {

int fd = open(device, O\_RDWR | O\_NOCTTY | O\_NDELAY);

if (fd == -1) {

std::cerr << "Unable to open serial port: " << strerror(errno) << std::endl;

return -1;

}

struct termios options;

tcgetattr(fd, &options);

cfsetispeed(&options, B9600);

cfsetospeed(&options, B9600);

options.c\_cflag |= (CLOCAL | CREAD);

options.c\_cflag &= ~PARENB;

options.c\_cflag &= ~CSTOPB;

options.c\_cflag &= ~CSIZE;

options.c\_cflag |= CS8;

tcsetattr(fd, TCSANOW, &options);

return fd;

}

void setSerialPortTimeouts(int fd, int readTimeoutMs, int writeTimeoutMs) {

struct termios options;

tcgetattr(fd, &options);

// Read timeout

options.c\_cc[VMIN] = 0; // Minimum number of characters to read

options.c\_cc[VTIME] = readTimeoutMs / 100; // Timeout in deciseconds

// Apply the settings

tcsetattr(fd, TCSANOW, &options);

}

uint16\_t modbusCRC(uint8\_t \*buffer, int length) {

uint16\_t crc = 0xFFFF;

for (int pos = 0; pos < length; pos++) {

crc ^= buffer[pos]; // XOR byte into least sig. byte of crc

for (int i = 8; i != 0; i--) { // Loop over each bit

if ((crc & 0x0001) != 0) { // If the LSB is set

crc >>= 1; // Shift right and XOR 0xA001

crc ^= 0xA001;

} else { // Else LSB is not set

crc >>= 1; // Just shift right

}

}

}

return crc;

}

C++

Copy code

**Summary**

This provided complete example shows how to:

1. Open and configure a serial port for communication.
2. Write to and read from MODBUS holding registers using specified function codes.
3. Implement CRC checks to ensure data integrity.

Each function deals with important aspects such as handling serial port communication, framing the MODBUS message, and performing CRC checks.

If you have any further questions or need additional functionality, feel free to ask!