# HopeRF


**Mike McCauley**

**Documentation for the HopeRF 1.2 communications library for Arduino.**

## 1.0 Introduction

Arduino is a low cost microcontroller with Open Source hardware, see http://www.arduino.cc. HopeRF is a communications library for Arduino that allows multiple Arduino's to communicate using low-cost 'transparent' serial RF data transceivers, such as the HM-TR from HopeRF (http://www.hoperf.com).

The document describes the HopeRF library and how to install and use it. Detailed API documentation can be found at
http://www.open.com.au/mikem/arduino/HopeRF/index.html

## 2.0 Overview

HopeRF is an Arduino library that provides features to send short messages between nodes equipped with RF serial data transceivers. It consists of several classes to provide various levels of features:

- Basic unaddressed, unreliable messages with 16 bit Frame Check Sequence (checksum).
- Addressed, unreliable messages for point-to-point communications in a network or mesh.
- Addressed, reliable messages for point-to-point communications in a network or mesh.. This is reliable in the sense that it provides positive acknowledgement of each message, with timeouts and retransmissions.

It is intended to be compatible with the HopeRF HM-TR 433MHz FSK transceivers (although any 'transparent' serial data radio should work just as well). It uses the stan-

dard Arduino HardwareSerial class to communicate with the RF module serially through an on board UART on Arduino. This means you can have as many transceivers as you have serial port UARTs on your Arduino.

Runs on Arduino Diecimila, Arduino Mega and possibly others.

The HopeRF library also compiles on Linux, and the `linux` directory in the distruibution contains Makefile, sample programs and a test suite, which compile and run on Linux. The server program talks to a HM-TR over a serial port at /dev/ttyUSB0.

## 3.0 Supported hardware.

This library is expected to work with any ‘transparent’ data radio transceiver, however it was developed and tested with the HopeRF HM-TR transceivers.

### 3.1 Hope HM-TR

The HopeRF HM-TR transceiver is an inexpensive 433MHz ’transparent’ serial transceiver. It handles internally all the issues of preamble, synchronisation etc (which is the bulk of the work in my otherArduino library, VirtualWire). Although the HM-TR is more expensive than the bare 433 MHz transceivers supported by VirtualWire, it requires much fewer compute resources from the Arduino (in fact the HM-TR has an ATMega on it specifically to do the serial-433MHz translation). However the HM-TR does not have any error detection built in. That is provided by this library.

The test devices were obtained from MicroZed conputers (http://www.microzed.com.au). We bought a package of 2 433MHz devices, including antennas for AUD45.00 (about US$35).

These transceivers have 6 connections:

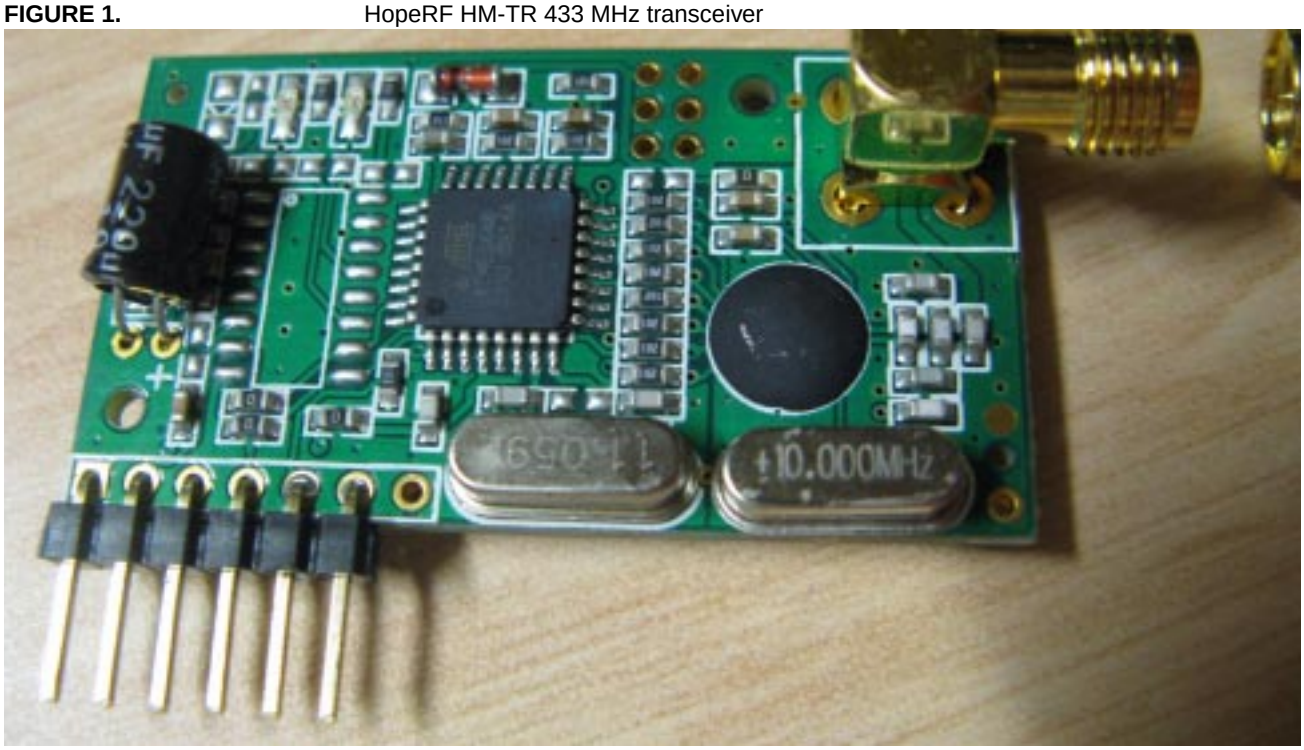
- VDD (5V)
- DTX Data transmit from the module
- Ground
- DRX Data input to the module
- CONFIG
- ENABLE

They are available in 2 flavours with either TTL or RS232 interfaces.

According to Silicon Chip magazine, the HM-TR has a good quality transmitter. It can be extensively configured via the serial port, including frequency, power, data rate etc. Unfortunately, configuration must be done by setting CONFIG high at startup time, then using the HopeRF configuration program (Windows only) to set the configuration. That process is not covered here.

The ENABLE pin enables the transceiver. If ENABLE is low, the transceiver goes into a low power sleep mode. The HopeRF library does not use or require the ENABLE or CONFIG pins.

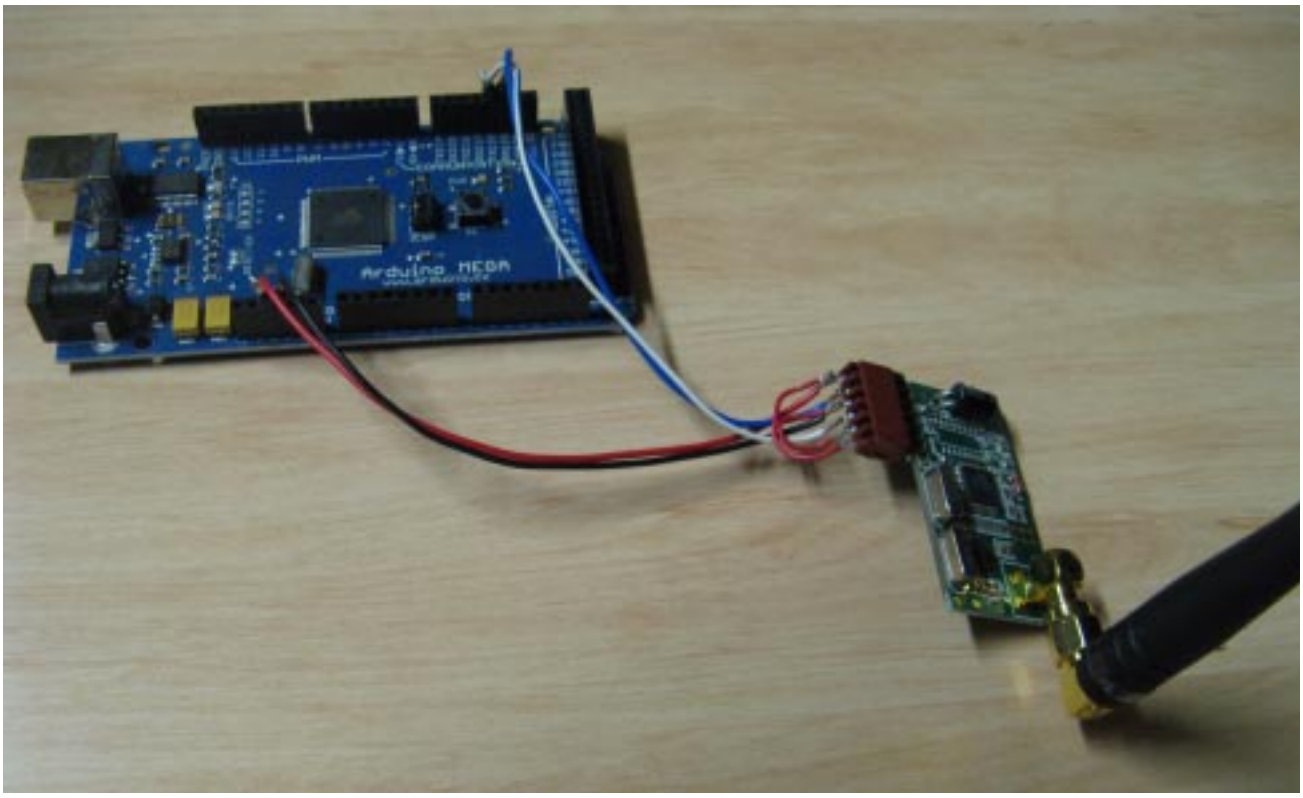
**FIGURE 1.** HopeRF HM-TR 433 MHz transceiver




Details at http://www.hoperf.com/pro/HM-TR.html

These devices are easy to connect to an Arduino: they require only +5V, GND and TTL transmit and receive pins. It can connect directly to Arduino serial port UART pins, and use the 5V power from the Arduino. Only 4 interconnection wires are required.

**FIGURE 2.** HM-TR connected to Arduino Mega




## 4.0 Downloading and installation

The latest version of this document is available from

`http://www.open.com.au/mikem/arduino/HopeRF/HopeRF.pdf`

Download the HopeRF distribution from

`http://www.open.com.au/mikem/arduino/HopeRF/HopeRF-1.2.zip`

To install, unzip the library to a sub-directory of the `hardware/libraries` sub-directory of your Arduino application directory. Then launch the Arduino environment; you should see the library in the Sketch->Import Library menu, and example code in File->Sketchbook->Examples->Library-HopeRF menu.

The complete documentation of the library API can be found at:

`http://www.open.com.au/mikem/arduino/HopeRF/index.html`

# 5.0 Sample code

The following samples are available as examples in the HopeRF distribution.

## 5.1 HRFMessage

### 5.1.1 HRFMessageClient

Sends an unaddressed message every second. Any HRFMEssageServer within range will receive the message and send a reply. Test this with the HRFMessageServer below.

```
#include <HRFMessage.h>
// Declare the HRFMessage to use Serial1 for IO
// with the HM-TR module
HRFMessage client(&Serial1);
long      lastSendTime = 0;
void setup()
{
    Serial.begin(9600);  // Monitoring via USB
    Serial1.begin(9600); // defaults to 9600
}
void loop()
{
  // Send a message to the server at most once a second
  long thisTime = millis();
  if (thisTime > lastSendTime + 1000)
  {
    client.send((uint8_t*)"test\n", 6);
    Serial.print("sending\n");
    lastSendTime = thisTime;
  }
  // But always look for a reply
  uint8_t buf[HRF_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);
  if (client.recv((uint8_t*)&buf, &len))
  {
    // Got a reply from the server
    Serial.print("got: ");
    Serial.print((const char*)buf);
  }
}
```

### 5.1.2 HRFMessageServer

Server to work with HRFMessageClient above. When it receives a message from HRFMessageClient it sends a reply.

```
#include <HRFMessage.h>
// Declare the HRFMessage to use Serial1 for
// IO with the HM-TR module
HRFMessage server(&Serial1);
void setup()
{
    Serial.begin(9600);  // Monitoring via USB
    Serial1.begin(9600); // defaults to 9600
```

```
}
void loop()
{
    uint8_t buf[HRF_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (server.recv((uint8_t*)&buf, &len))
    {
      // Got a message from the client.
      // Send a reply back
      server.send((uint8_t*)"reply\n", 7);
      Serial.print("got: ");
      Serial.print((const char*)buf);
    }
}
```

## 5.2 HRFDatagram

### 5.2.1 HRFDatagramClient

Implements a simple wireless client. Sends an addressed message to another Arduino running the HRFDatagramServer code below and gets a reply.

```
#include <HRFDatagram.h>
// Declare the HRFDatagram to use Serial1 for
// IO with the HM-TR module
// The address of this node is 10.
HRFDatagram client(&Serial1, 10);
long      lastSendTime = 0;
void setup()
{
    Serial.begin(9600);  // Monitoring via USB
    Serial1.begin(9600); // defaults to 9600
}
void loop()
{
  // Send a message to the server at most once a second
  long thisTime = millis();
  if (thisTime > lastSendTime + 1000)
  {
    // Send a message to node 11 (the server)
    client.sendto(11, (uint8_t*)"test\n", 6);
    Serial.print("sending\n");
    lastSendTime = thisTime;
  }
  // But always look for a reply
  uint8_t from;
  uint8_t buf[HRF_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);
  if (client.recvfrom((uint8_t*)&buf, &len, &from))
  {
    // Got a reply from the server
    Serial.print("got from node: ");
    Serial.print(from, DEC);
    Serial.print(": ");
```

```
    Serial.print((const char*)buf);
  }
}
```

#### 5.2.2 HRFDatagramServer

Implements a simple wireless server. Waits for a message from another Arduino running the HRFDatagramClient code above and sends a reply.

```
#include <HRFDatagram.h>
// Declare the HRFMessage to use Serial1 for
// IO with the HM-TR module
// The address of this node is 11.
HRFDatagram server(&Serial1, 11);
void setup()
{
    Serial.begin(9600);  // Monitoring via USB
    Serial1.begin(9600); // Defaults to 9600
}
void loop()
{
    uint8_t from;
    uint8_t buf[HRF_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (server.recvfrom((uint8_t*)&buf, &len, &from))
    {
      // Got a message from the client.
      // Send a reply back to whomever sent it
      server.sendto(from, (uint8_t*)"reply\n", 7);
      Serial.print("got from node: ");
      Serial.print(from, DEC);
      Serial.print(": ");
      Serial.print((const char*)buf);
    }
}
```

### 5.3 HRFReliableDatagram

#### 5.3.1 HRFReliableDatagramClient

Implements reliable messages between client and server. Sends a message every second to HRFReliableDatagramServer below. Waits for an acknowledgement from the server, retransmitting if necessary. Thern possibly receives a reply message from the server, which is automatically acknowledged to the server

```
#include <HRFReliableDatagram.h>
// Declare the HRFDatagram to use Serial1 for
//IO with the HM-TR module
// The address of this node is 10.
HRFReliableDatagram client(&Serial1, 10);
long      lastSendTime = 0;
void setup()
{
    Serial.begin(9600);  // Monitoring via USB
    Serial1.begin(9600); // Defaults to 9600
```

```
   client.setTimeout(500);
}
void loop()
{
  // Send a message to the server at most once a second
  long thisTime = millis();
  if (thisTime > lastSendTime + 1000)
  {
    // Send a message to node 11 (the server)
    Serial.print("sending\n");
    if (!client.sendtoWait(11, (uint8_t*)"test\n", 6))
      Serial.print("sendtoWait failed\n");
    lastSendTime = thisTime;
  }
  // But always look for a reply
  uint8_t from;
  uint8_t buf[HRF_MAX_MESSAGE_LEN];
  uint8_t len = sizeof(buf);
  if (client.recvfromAck((uint8_t*)&buf, &len, &from))
  {
     // Got a reply from the server
    Serial.print("got from node: ");
    Serial.print(from, DEC);
    Serial.print(": ");
    Serial.print((const char*)buf);
  }
}
```

### 5.3.2 HRFReliableDatagramServer

```
#include <HRFReliableDatagram.h>
// Declare the HRFMessage to use Serial1 for
// IO with the HM-TR module
// The address of this node is 11.
HRFReliableDatagram server(&Serial1, 11);
void setup()
{
    Serial.begin(9600);  // Monitoring via USB
    Serial1.begin(9600); // Defaults to 9600
   server.setTimeout(500);
}
void loop()
{
    uint8_t from;
    uint8_t buf[HRF_MAX_MESSAGE_LEN];
    uint8_t len = sizeof(buf);
    if (server.recvfromAck((uint8_t*)&buf, &len, &from))
    {
      Serial.print("got from node: ");
      Serial.print(from, DEC);
      Serial.print(": ");
      Serial.print((const char*)buf);
      // Got a message from the client.
      // Send a reply back to whomever sent it
```

```
      if (!server.sendtoWait(from, (uint8_t*)"reply\n", 7))
       Serial.print("sendtoWait failed\n");
    }
}
```

## 6.0 Implementation Details

The code is a set of conventional C++ classes, with no interrupt service routines (although the HardwareSerial class which HopeRF uses does use interrupts to receive each serial character from the transceiver).

HopeRF uses the Arduino HardwareSerial class to implement the serial IO with the transceiver module.

### 6.0.1 HRFMessage

HRFMessage can send messages of up to HRF_MAX_PAYLOAD (61) octets.

Each message is transmitted as:

- 1 octet of message length (4 to 64), count includes length octet and FCS octets
- message (up to 61 octets)
- 2 octets FCS, sent low byte-hi byte

FCS (16 bits) is the complement of CCITT CRC-16 of all octets in the message, including the length octet, but not including the FCS.

### 6.0.2 HRFDatagram

HRFDatagram can send messages of up to HRF_MAX_DATAGRAM_PAYLOAD (59) octets, using 8 bit node addresses. Broadcast address of HRF_BROADCAST_ADDRESS (0xFF) can be used to send a message to all nodes within range.

Each message is transmitted as:

- 1 octet of message length (4 to 64), count includes length octet and FCS octets
- 1 octet of DEST address
- 1 octet of SRC address
- message (up to 59 octets)
- 2 octets FCS, sent low byte-hi byte

### 6.0.3 HRFReliableDatagram

HRFReliableDatagram can send messages of up to HRF_MAX_RELIABLE_DATAGRAM_PAYLOAD (58) octets using 8 bit node addresses.

When a message is received by a node, it is automatically acknowledged. When a message is transmitted by a node, it automcatically waits until an acknowledgement is received, retransmitting as required until an acknowledgement is received or all retries are exhausted. Duplicate messages (due to lost acknowledgements) are dropped. New messages received while waiting for an acknowledgement are ignored. Broadcast messages are never acknowledged.

Each message is transmitted as:

- 1 octet of message length (4 to 64), count includes length octet and FCS octets
- 1 octet of DEST address
- 1 octet of SRC address
- 1 octet of FLAGS/SQN. The most significant bit is an acknowledge flag (set in acknowledge messages), and the lower 7 bits are a message sequence number.
- message (up to 58 octets)
- 2 octets FCS, sent low byte-hi byte

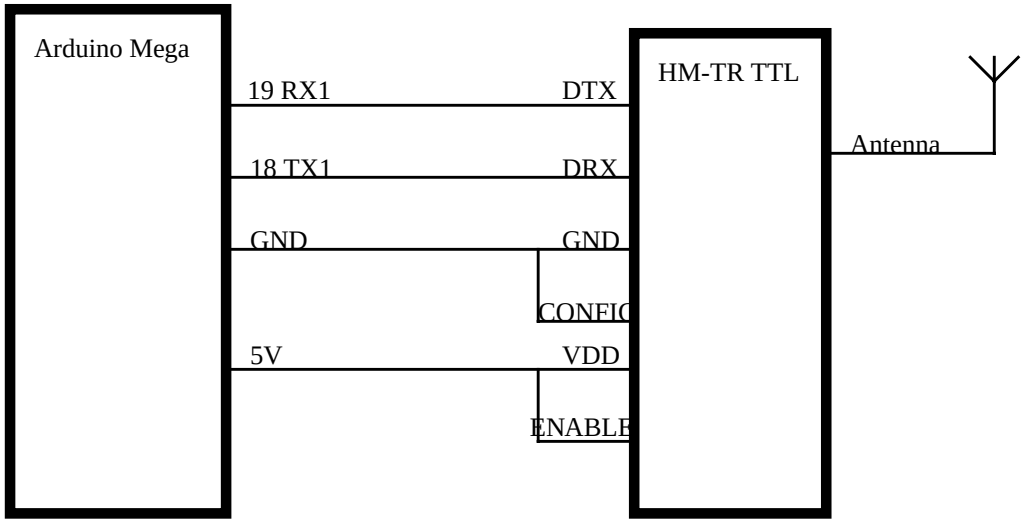
## 7.0 Performance

Tests were performed with 2 Arduino Mega and HM-TR with the supplied 433MHz omnidirectional antenna. Transmitter and receiver were approx. 1m above ground, with the test conducted line of sight over a bitumen road. The software running was the HRFReliableDatagram Client and Server above. The HM-TR transmit and receive lights were monitored to determine when errors and retransmissions occurred. At about 200m, occasional retransmissions were seen. With suitable retries settings larger distances of the order of 300m (as per the HopeRF literature) should be achievable.

# 8.0 Connections

## 8.1 HM-TR transceiver

**FIGURE 3.** Wiring for HM-TR receiver to Arduino Mega




Note that in the HM-TR ENABLE is high, meaning the HM-TR is always enabled. CONFIG is low, meaning operate (not configure).

# 9.0 Copyright and License



## 9.1 Open Source Licensing GPL V2

This is the appropriate option if you want to share the source code of your application with everyone you distribute it to, and you also want to give them the right to share who uses it. If you wish to use this software under Open Source Licensing, you must contribute all your source code to the open source community in accordance with the GPL Version 2 when your application is distributed. See http://www.gnu.org/copyleft/gpl.html

### 9.2 Commercial Licensing

This is the appropriate option if you are creating proprietary applications and you are not prepared to distribute and share the source code of your application. Contact info@open.com.au for details.