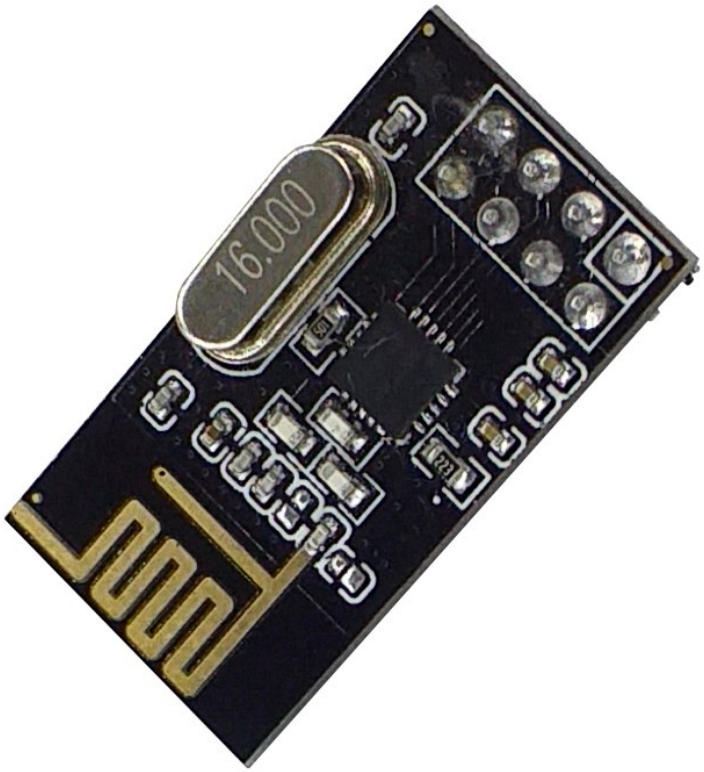
Welcome!

Thank you very much for purchasing our AZ-Delivery nRF24L01 module. On the following pages, we will introduce you to how to use and setup this handy device.

Have fun!



Many projects require communication between two devices. Communications are in most cases via cables, and some communication modules are able to communicate with each other wirelessly. So with these wireless communication modules we are able to create a wireless network of devices sending data from one end to the other. These modules are mostly used in devices that monitor sensor data, control robots, home automation, etc.

nRF24L01 is one of these wireless modules. It uses radio frequencies to communicate with another nRF24L01 module, two way communication. nRF24L01 is super cheap, and very small in size enabling you to interface it into almost all of your projects.

The nRF24L01 module operates in 2.4GHz ISM frequency band and uses GFSK modulation for data transmission where data transfer rates are one of the following 250kbps, 1Mbps and 2Mbps.

2.4GHz band is one of the Industrial, Scientific and Medical bands, short ISM bands, reserved itrenationally for the use of unlicensed low-powered devices. Examples of these devices are mobile phones, bluetooth devices, near field communication (NFC) devices, or wireless computer networks (WiFis) all use the ISM frequencies.

Specifications

Frequency Range:	2.4GHz, ISM Band
Maximum Air Data Rate:	2Mb/s
Modulation Format:	GFSK
Maximum Output Power:	0dBm
Operating Supply Voltage:	1.9V to 3.6V
Maximum Operating Current:	13.5mA
Min. Current (Standby Mode):	26µA
Logic Inputs:	5V tolerant
Communication Range:	100+ meters (line of sight)

The operating voltage range of the module is from 1.9V to 3.6V. The logic pins are 5V tolerant, so we can easily connect it to an Arduino or any 5V logic microcontroller without using any logic level converter.

The module supports programmable output power in one of the following ranges 0dBm, -6dBm, -12dBm or -18dBm.

The module consumes around 12mA during transmission at 0dBm, which is even lower than a single LED. In standby mode it consumes 26µA and at power down mode 900nA. That’s why they’re almost the best wireless device for low-power applications.

The module uses on board antenna. This allows for a more compact board of the module. However, the smaller antenna also means a lower transmission range, which is around 100m, for our version of module. This 100m is transmission range of the module outdoors in open space. The range indoors, especially through walls, will be slightly weakened.

SPI interface

The module uses SPI interface to communicate with microcontrollers. This means that the module communicates over 4 wires with maximum data rate of 10Mbps. All the parameters such as frequency channel (125 selectable channels), output power (0dBm, -6dBm, -12dBm or -18dBm), and data rate (250kbps, 1Mbps, or 2Mbps) can be configured by microcontroller through SPI interface.

The SPI bus uses a concept of a Master and Slaves, in most common applications where our Arduino is the Master and the nRF24L01 module is the Slave. Unlike the I2C bus the number of slaves on the SPI bus is limited, on the Arduino Uno you can use a maximum of two SPI slaves i.e. two nRF24L01 modules.

Working principle

The module transmits and receives data on a certain frequency called channel. In order for two or more transceiver modules to communicate with each other, they need to be on the same channel. This channel could be any frequency in the 2.4GHz ISM band or to be more precise, it could be a frequency between 2.400GHz and 2.525 GHz (which is 2400 to 2525 MHz).

Each channel occupies a bandwidth of less than 1MHz. This gives us 125 possible channels with 1MHz spacing. So, the module can use 125 different channels which gives a possibility to have a network of 125 independently working modems in one place.

One channel occupies a bandwidth of less than 1MHz at 250kbps and 1Mbps air data rate. However at 2Mbps air data rate, 2MHz bandwidth is occupied (wider than the resolution of RF channel frequency setting). So, to ensure non-overlapping channels and reduce cross-talk in 2Mbps mode, you need to keep 2MHz spacing between two channels.

RF channel frequency of your selected channel is set according to the following formula: Freq (Selected) = 2400 + CH (Selected)
For example, if you select 108 as your channel for data transmission, the RF channel frequency of your channel would be 2508MHz (2400 + 108).

Multiceiver Network

The module provides a feature called Multiceiver. It's an abbreviation for Multiple Transmitters Single Receiver. This means that each RF channel is logically divided into 6 parallel data channels called Data Pipes. In other words, a data pipe is a logical channel in the physical RF Channel. Each data pipe has its own physical address (Data Pipe Address) and can be configured.

Here we have the primary receiver acting as a hub receiver collecting information from 6 different transmitter nodes simultaneously. The hub receiver can stop listening any time and acts as a transmitter. But this can only be done one pipe/node at a time.

Enhanced ShockBurst Protocol

This module uses a packet structure known as Enhanced ShockBurst. This simple packet structure is broken down into 5 different fields:

- Preamble
- Address
- Packet Control Field (PCF)
  - payload length
  - packet ID
  - no acknowledge
- Payload
- Cyclic Redundancy Check (CRC)

The original ShockBurst structure consisted only of Preamble, Address, Payload and the Cyclic Redundancy Check (CRC) fields. Enhanced ShockBurst brought greater functionality for more enhanced communications using a newly introduced Packet Control Field (PCF).

This new structure is great for a number of reasons. Firstly, it allows for variable length payloads with a payload length specifier, meaning payloads can vary from 1 to 32 bytes. Secondly, it provides each sent packet with a packet ID, which allows the receiving device to determine whether a message is new or whether it has been retransmitted (and thus can be ignored). Finally, and most importantly, each message can request an acknowledgement to be sent when it is received by another device.

Automatic Packet Handling

Let's explain three scenarios so we can get a better understanding of how two modules transact with each other.

First a transaction with acknowledgement and interrupt: The transmitter starts a communication by sending a data packet to the receiver. Once the whole packet is transmitted, transmitter waits (around 130µs) for the acknowledgement packet (ACK packet) to be sent back for receiver. When the receiver receives the packet, it sends an ACK packet to the transmitter. When the transmitter receive the ACK packet it assert it by generating the interrupt signal (on IRQ pin) to indicate the new data is available.

Second is a transaction with data packet lost: This is a negative scenario where a retransmission is needed due to loss of the packet transmitted. After the packet is transmitted, the transmitter waits for the ACK packet to receive. If the transmitter doesn't get it within Auto-Retransmit-Delay (ARD) time, the packet is retransmitted. When the retransmitted packet is received by the receiver, the ACK packet is transmitted back to transmitter which in turn generates interrupt at the transmitter.

Third a transaction with acknowledgement lost: This is again a negative scenario where a retransmission is needed due to loss of the ACK packet. Here even if the receiver receives the packet in the first attempt, due to the loss of ACK packet, transmitter thinks the receiver has not got the packet at all. So, after the Auto-Retransmit-Delay time is over, it retransmits the packet. Now when the receiver receives the packet containing the same packet ID as previous, it discards it and sends an ACK packet again.

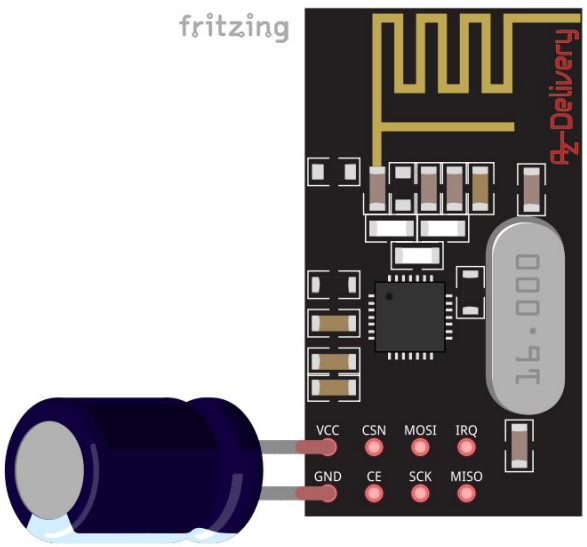
This whole packet handling is done automatically by the nRF24L01 chip without involvement of the microcontroller.

Improving the range of the Module

A key parameter for any wireless communication system is the communication range. In many cases it's the main factor for choosing an RF solution. Here are some examples of how we can improve the range for out module.

1. Reduce Power Supply Noise

Every RF circuit that generates a Radio Frequency (RF) signal, is very sensitive to power supply noise. If the power supply noise isn't controlled, it can significantly reduce the range you can get. Unless the power source is a stand-alone battery, there is a good chance that there is noise associated with the generation of the power. To prevent this noise from entering the system, it is advised to place a 10µF filter capacitor between the VCC and GND pins, and as physically close to the nRF24L01 module as possible, like on image below (watch for polarity of capacitor!).



2. Change your channel frequency

Another potential source of noise for an RF circuit is the outside environment, especially if you have neighboring networks set on the same channel or interference from other electronics. To prevent these signals from causing issues, we suggest using the highest 25 channels your module. Reason for this is because WiFi uses most of the lower channels.

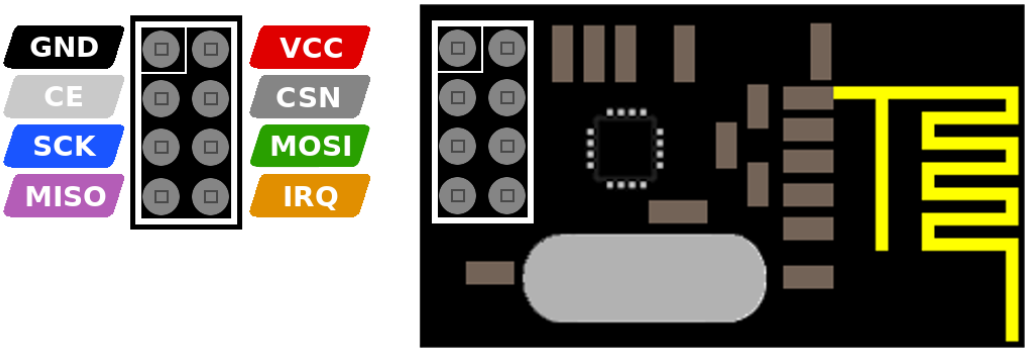
3. Lower Data Rate

The nRF24L01 module offers highest receiver sensitivity at 250Kbps speed which is -94dBm. However at 2MBps data rate, the receiver sensitivity drops to -82dBm. From this we can conclude that the receiver at 250Kbps is nearly 10 times more sensitive than at 2Mbps. That means the receiver can decode a signal that is 10 times weak. What does Receiver (Rx) sensitivity mean? Receiver sensitivity is the lowest power level at which the receiver can detect an RF signal. The larger the absolute value of the negative number, the better the receiver sensitivity. For example, a receiver sensitivity of −94dBm is better than a receiver sensitivity of −82dBm by 12dB. So, lowering the data rate can significantly improve the range you can achieve. Also, for most of your projects, 250Kbps speed is more than sufficient.

4. Higher Output Power

Setting maximum output power can also improve the communication range. The nRF24L01 module lets you choose one of the output power from following values 0dBm (max), -6dBm, -12dBm or -18dBm (min). Selecting 0dBm output power sends a stronger signal over the air.

The pinout of nRF24L01



GND is the Ground Pin. It is usually marked by encasing the pin in a square so it can be used as a reference for identifying the other pins.

VCC supplies power for the module, from 1.9V to 3.9V. You can connect it to 3.3V output from your Arduino.

CE (Chip Enable) is an active HIGH pin. When selected the nRF24L01 will either transmit or receive, depending upon which mode it is currently in.

CSN (Chip Select Not) is an active LOW pin and is normally kept HIGH. When this pin goes LOW, the nRF24L01 begins listening on its SPI port for data and processes it accordingly.

SCK (Serial Clock) accepts clock pulses provided by the SPI bus Master.

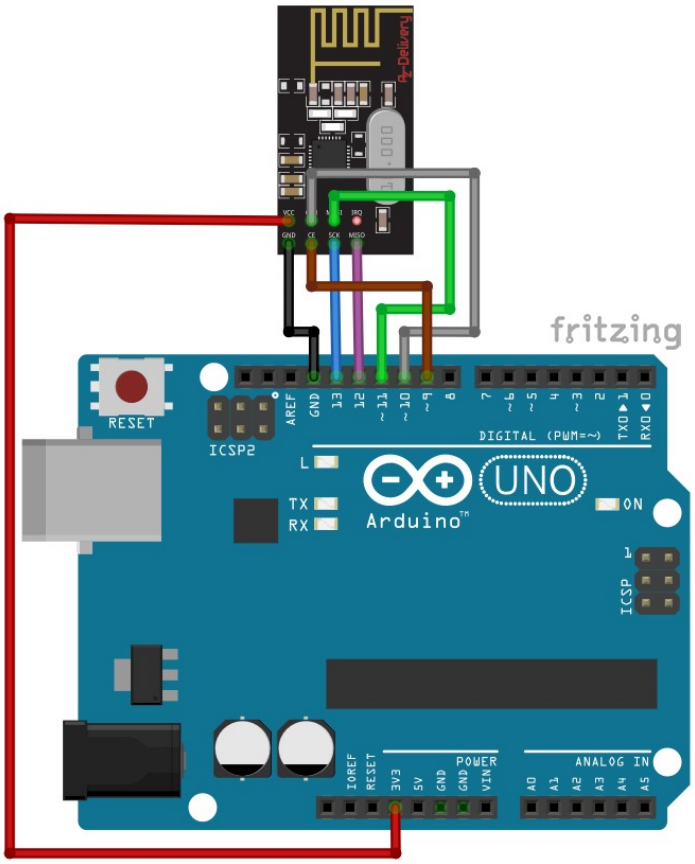
MOSI (Master Out Slave In) is SPI input to the nRF24L01.

MISO (Master In Slave Out) is SPI output from the nRF24L01.

IRQ is an interrupt pin that can alert the master when new data is available to process.

Remember connecting VCC to 5V pin will likely destroy your nRF24L01 module!

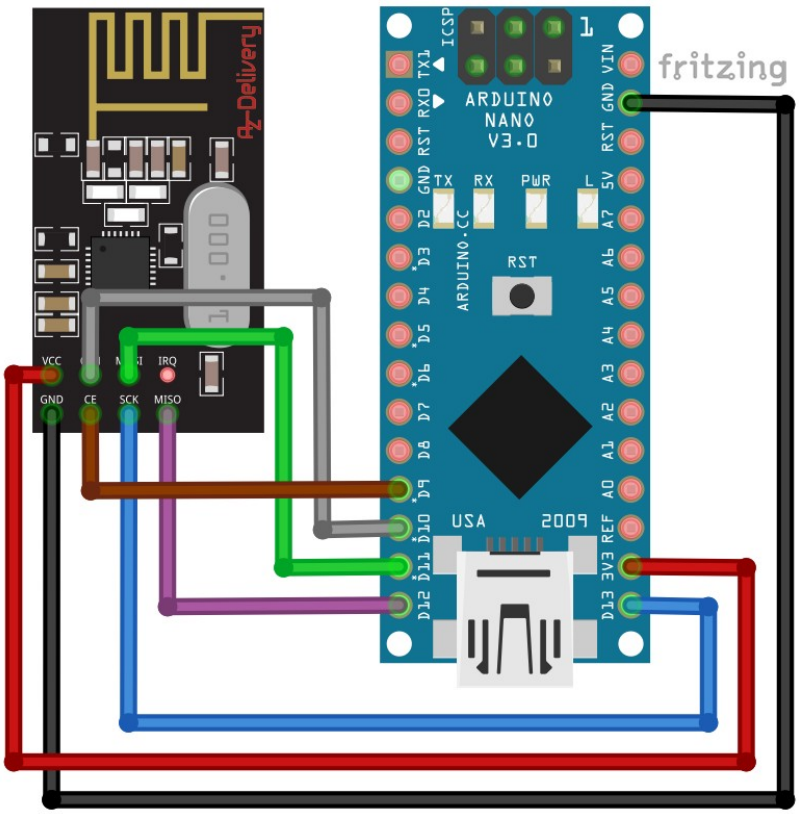
Connecting the module with the Arduino Uno



nRF24L01 pin	**>**	**Uno pin**	
VCC	>	3.3V	**Red wire**
GND	>	GND	**Black wire**
CSN	>	D10	**Gray wire**
CE	>	D9	**Brown wire**
MOSI	>	D11	**Green wire**
SCK	>	D13	**Blue wire**
IRQ	>	not connected	
MISO	>	D12	**Purple wire**

The pins CSN and CE can be connected to any digital pin on the Arduino. In our case, it's connected to digital pin 10 and 9 respectively.

Connecting the module with the Arduino Nano



nRF24L01 pin	**>**	**Nano pin**	
VCC	>	3V3	**Red wire**
GND	>	GND	**Black wire**
CSN	>	D10	**Gray wire**
CE	>	D9	**Brown wire**
MOSI	>	D11	**Green wire**
SCK	>	D13	**Blue wire**
IRQ	>	not connected	
MISO	>	D12	**Purple wire**

The pins CSN and CE can be connected to any digital pin on the Arduino. In our case, it's connected to digital pin 10 and 9 respectively.

Library for Arduino IDE

If you don’t have an Arduino IDE go on the link, download it and install it. Installation is really simple, just follow the instructions in the installalation wizard.

One of the popular libraries is RF24. This library has been around for several years. It is simple to use for beginners, but yet offers a lot for advanced users. You can download the latest version of the library on the RF24 GitHub. When you download it you will have a zip file, and to install it, open the Arduino IDE, go to *Sketch* > *Include Library* > *Add .ZIP Library*, and then select the zip file that you just downloaded.

We need to make two of these circuits, one for the transmitter and the other for the receiver. The wiring for both is indetical.

Simple example code for transmitter

In this example we will just send a "*Hello World*" message from the transmitter to the receiver. Here is the sketch for transmitter:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(9, 8);  // create an RF24 object, CE, CSN
const byte address[6] = "00001";  // address
void setup() {
   radio.begin();
   radio.openWritingPipe(address);  // set the address
   radio.stopListening();  // set module as transmitter
}
void loop() {
   const char text[] = "Hello World";  // message
   radio.write(&text, sizeof(text));  // send message
   delay(1000);
}
```

The sketch starts by including the libraries. *SPI.h* library handles the SPI communication while *nRF24L01.h* and *RF24.h* controls the module.

Next, we need to create an *RF24* object. The object takes two pin numbers as parameters to which signals *CE* and *CSN* are connected.

```
RF24 radio(CE, CSN);
```

Next we need to create a byte array which will represent the pipe address through which two modules communicate.

```
const byte address[6] = "00001";
```

We can change the value of this address to any 5-letter string such as "*node1*". The address is necessary if you have a few modules in a network. Thanks to the address, you can choose a particular module to which you are interested in communicating, so in our case we will have the same address for both the transmitter and the receiver.

In the setup function, we need to initialize the radio object using `radio.begin()` and using the `radio.openWritingPipe(address)` function. We set the address of the transmitter. We use the `radio.stopListening()` function which sets module as transmitter.

In the loop section, we create an array of characters to which we assign the message "*Hello World*". Using the `radio.write()` function we will send that message to the receiver. The first argument here is the message that we want to send. The second argument is the number of bytes present in that message: `radio.write(&text, sizeof(text));`
Through this method, you can send up to 32 bytes at a time. Because that is the maximum size of a single packet nRF24L01 can handle. If you need a confirmation that the receiver received data, the method `radio.write()` returns a bool value. If it returns TRUE, the data reached the receiver. If it returns FALSE, the data has been lost.

One thing that you should remember, the radio.write() function blocks the program until it receives the acknowledgment or runs out of all attempts of retransmission.

Simple example code for receiver

Here is the sketch for our receiver:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
RF24 radio(9, 8);  // CE, CSN
const byte address[6] = "00001";  // address
void setup() {
   while(!Serial);
      Serial.begin(9600);
   radio.begin();
   radio.openReadingPipe(0, address); // set the address
   radio.startListening();  // set module as receiver
}
void loop() {
   // Read the data if available in buffer
   if(radio.available()) {
      char text[32] = {0};
      radio.read(&text, sizeof(text));
      Serial.println(text);
   }
}
```

Most of this sketch is the same as a sketch for transceiver module. Here are the differences:

At the beginning of the setup function we start the serial communication. We use this so that we can show the received message in the Serial Monitor.

Next using radio.setReadingPipe() function we set the same address as transmitter and in that way we enable communication between transmitter and receiver. `radio.openReadingPipe(0, address)`
The first argument is the number of the stream. You can create up to 6 streams that respond to different addresses. We created only address for the stream number 0. The second argument is the address to which the stream will react to collect the data.

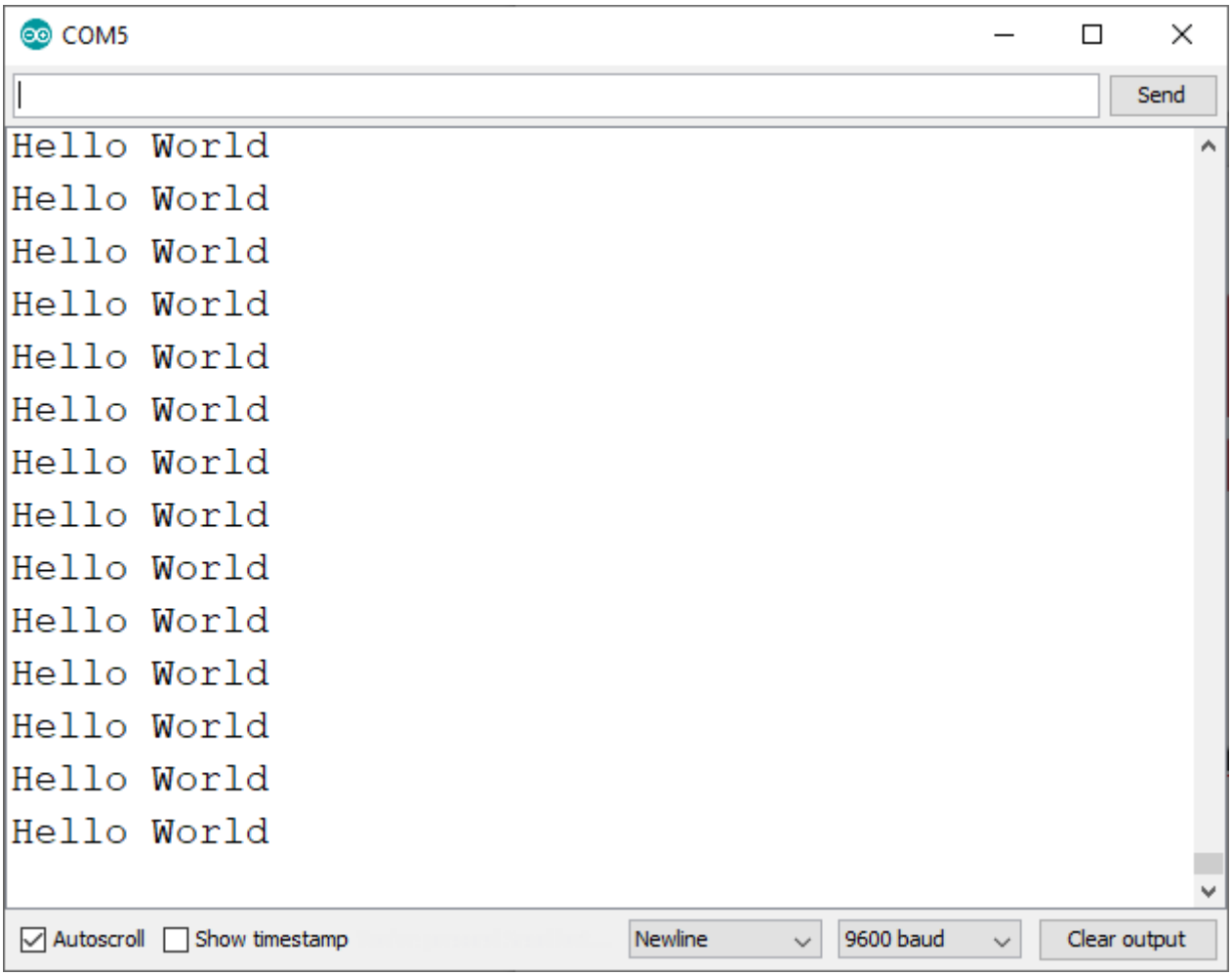
The next step is to set the module as a receiver and start receiving data. To do that we use `radio.startListening()` function. From that moment the modem waits for data sent to the specified address.

In the loop function we check whether any data has arrived at the address using `radio.available()` method. This method returns TRUE value if we any data is available in buffer.

```
if (radio.available()) {
   char text[32] = {0};
   radio.read(&text, sizeof(text));
   Serial.println(text);
}
```

If the data is received, then it creates an array of 32 characters filled with zeros (later the program will fill it with the received data). To read the data we use the method `radio.read(&text, sizeof(text))`. This will store the received data in to our character array.

At the end we just print the received message in Serial Monitor. If you did everything ok and there are no mistakes in connections, you should see message in your Serial Monitor, like this:



There are three other parameters that we can set between `radio.begin` and `radio.openReadingPipe` (or `openWrittingPipe`). In our example sketches these parameters are at default value.

First is `radio.setChannel(value)`, where the "*value*" is the channel number in the range from 0 to 125. In our sketch we used channel 0, in `openReadingPipe(0, address)` function.

Second is `radio.setPALevel(value)` where the "*value*" is one from the following values:

» `RF24_PA_MIN` = -18dBm (default)
» `RF24_PA_LOW` = -12dBm
» `RF24_PA_MED` = -6dBm
» `RF24_PA_HIGH` = 0dBm

With this we setup the power amplifier level used by module.

Third is `radio.setDataRate(value)`, where the "*value*" is one from the following values:

» `RF24_250KBPS` for 250kbs (default)
» `RF24_1MBPS` for 1Mbps
» `RF24_2MBPS` for 2Mbps

With this we setup data transmission rate.

You've done it, you can now use your module for your projects.

Now it is time to learn and make the Projects on your own. You can do that with the help of many example scripts and other tutorials, which you can find on the internet.

If you are looking for the high quality products for Arduino and Raspberry Pi, AZ-Delivery Vertriebs GmbH is the right company to get them from. You will be provided with numerous application examples, full installation guides, eBooks, libraries and assistance from our technical experts.

https://az-delivery.de

Have Fun!

Impressum

https://az-delivery.de/pages/about-us