* <https://tmrh20.github.io/RF24Mesh/>
* <https://github.com/nRF24/RF24Mesh/tree/master/examples>
* **DHCP**

|  |  |
| --- | --- |
|  |  |

Only to be used on the master node.

Provides automatic configuration for sensor nodes, similar to DHCP. Call immediately after calling network.update() to ensure address requests are handled appropriately

|  |
| --- |
| #include "RF24.h" |
|  |

|  |
| --- |
| #include "RF24Network.h" |
|  |

|  |
| --- |
| #include "RF24Mesh.h" |
|  |

|  |
| --- |
| #include <SPI.h> |
|  |

|  |
| --- |
| //#include <printf.h> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*\*\*\* Configure the nrf24l01 CE and CS pins \*\*\*\*/ |
|  |

|  |
| --- |
| RF24 radio(7, 8); |
|  |

|  |
| --- |
| RF24Network network(radio); |
|  |

|  |
| --- |
| RF24Mesh mesh(radio, network); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| /\*\* |
|  |

|  |
| --- |
| User Configuration: nodeID - A unique identifier for each radio. Allows addressing |
|  |

|  |
| --- |
| to change dynamically with physical changes to the mesh. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| In this example, configuration takes place below, prior to uploading the sketch to the device |
|  |

|  |
| --- |
| A unique value from 1-255 must be configured for each node. |
|  |

|  |
| --- |
| This will be stored in EEPROM on AVR devices, so remains persistent between further uploads, loss of power, etc. |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| \*\*/ |
|  |

|  |
| --- |
| #define nodeID 1 |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| uint32\_t displayTimer = 0; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| struct payload\_t { |
|  |

|  |
| --- |
| unsigned long ms; |
|  |

|  |
| --- |
| unsigned long counter; |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void setup() { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| Serial.begin(115200); |
|  |

|  |
| --- |
| //printf\_begin(); |
|  |

|  |
| --- |
| // Set the nodeID manually |
|  |

|  |
| --- |
| mesh.setNodeID(nodeID); |
|  |

|  |
| --- |
| // Connect to the mesh |
|  |

|  |
| --- |
| Serial.println(F("Connecting to the mesh...")); |
|  |

|  |
| --- |
| mesh.begin(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void loop() { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| mesh.update(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Send to the master node every second |
|  |

|  |
| --- |
| if (millis() - displayTimer >= 1000) { |
|  |

|  |
| --- |
| displayTimer = millis(); |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // Send an 'M' type message containing the current millis() |
|  |

|  |
| --- |
| if (!mesh.write(&displayTimer, 'M', sizeof(displayTimer))) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| // If a write fails, check connectivity to the mesh network |
|  |

|  |
| --- |
| if ( ! mesh.checkConnection() ) { |
|  |

|  |
| --- |
| //refresh the network address |
|  |

|  |
| --- |
| Serial.println("Renewing Address"); |
|  |

|  |
| --- |
| if(!mesh.renewAddress()){ |
|  |

|  |
| --- |
| //If address renewal fails, reconfigure the radio and restart the mesh |
|  |

|  |
| --- |
| //This allows recovery from most if not all radio errors |
|  |

|  |
| --- |
| mesh.begin(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } else { |
|  |

|  |
| --- |
| Serial.println("Send fail, Test OK"); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } else { |
|  |

|  |
| --- |
| Serial.print("Send OK: "); Serial.println(displayTimer); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| while (network.available()) { |
|  |

|  |
| --- |
| RF24NetworkHeader header; |
|  |

|  |
| --- |
| payload\_t payload; |
|  |

|  |
| --- |
| network.read(header, &payload, sizeof(payload)); |
|  |

|  |
| --- |
| Serial.print("Received packet #"); |
|  |

|  |
| --- |
| Serial.print(payload.counter); |
|  |

|  |
| --- |
| Serial.print(" at "); |
|  |

|  |
| --- |
| Serial.println(payload.ms); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

* **releaseAddress**

Releases the currently assigned address lease.

Useful for nodes that will be sleeping etc.

Nodes should ensure that addresses are releases successfully prior to renewal.

|  |  |  |  |
| --- | --- | --- | --- |
| * **write** | ( | uint16\_t | *to\_node*, |
|  |  | const void \* | *data*, |
|  |  | uint8\_t | *msg\_type*, |
|  |  | size\_t | *size* |
|  | ) |  |  |