• E-Mail: mike@pwpublishing.ltd.uk

UKHASnet

Mike Richards G4WNC introduces readers to a low-speed wireless-based mesh network that can be built easily and cheaply and has lots of potential applications.

Over the past few months I've been involved with the UK High Altitude Society and, in particular, their UKHASnet project. The UK High Altitude Society is a loose organisation of enthusiasts who work together to develop the technology to support all things to do with high altitude ballooning. I've mentioned them here before and I'm sure you will have heard of some of their record-breaking exploits. The group is a true network of hackers in the creative sense because they are using their engineering and programming skills to adapt and make the most of, cheap electronic components and modules. The UKHASnet project is a new networking system that was originally conceived to support high altitude ballooning but has a much wider appeal for any situation where you want to connect things wirelessly. Let's start with an overview of the UKHASnet network.

UKHASnet

You will doubtless have heard the phrase Internet of Things or IoT being generally overused by marketing people. However, when you try to use the technology, you will quickly find that adding wireless connectivity to your projects is not as easy or as cheap as it ought to be. Bluetooth and Wi-Fi modules tend to be very expensive and the Bluetooth licensing process severely penalises the development of low volume commercial products. The UKHASnet approach has been to define a simple new network that's ideal for handling low-speed data such as sensor readings and simple commands. In addition to providing simple connections, the UKHASnet has been designed to operate as a mesh network. By that I mean that repeaters can be used to extend the reach of the network in a mesh-like manner, **Fig. 1**.

At the heart of this network is the use of RF transceiver modules from Hope RF. The RFM69HW is the preferred module and this is a programmable, VHF/UHF transceiver with an output power of up to 100mW and frequency setting in 61Hz steps, **Fig. 2**. The unit is designed for operation in the sub-1GHz licence exempt bands and can be controlled using SPI commands from a local processor. In addition to being a very useful transceiver, the main attraction of the RFM69HW is the price, which can be as low as £3-£4 each in small quantities! To make a working node, some form of microcontroller is also required to read the sensor data and to control the RFM69HW. The Atmel ATmega328 as used by many of the Arduino boards is a popular choice. You can use a standard Arduino board, one of the many clones or even a bare ATMega328 chip. In the latter case, by loading an Arduino bootloader, you can program it using the popular Arduino IDE (Integrated Development Environment). As you will see, it is very easy to build UKHASnet nodes using stripboard construction but there are some other interesting options available, as I will show you later.

Inside UKHASnet

Before I move on to the practicalities of building nodes and making things work, let's have a closer look at the protocols used on the network. In order to send data successfully over a radio link, you need an organised way to manage the transmission. The solution used by UKHASnet is to wrap the data in packets. This technique is used extensively in modern applications ranging from our amateur radio packet networks through to the internet so it is well proven. In the UKHASnet, these packets contain information to wake up the receiver, synchronise the timing, send the data and

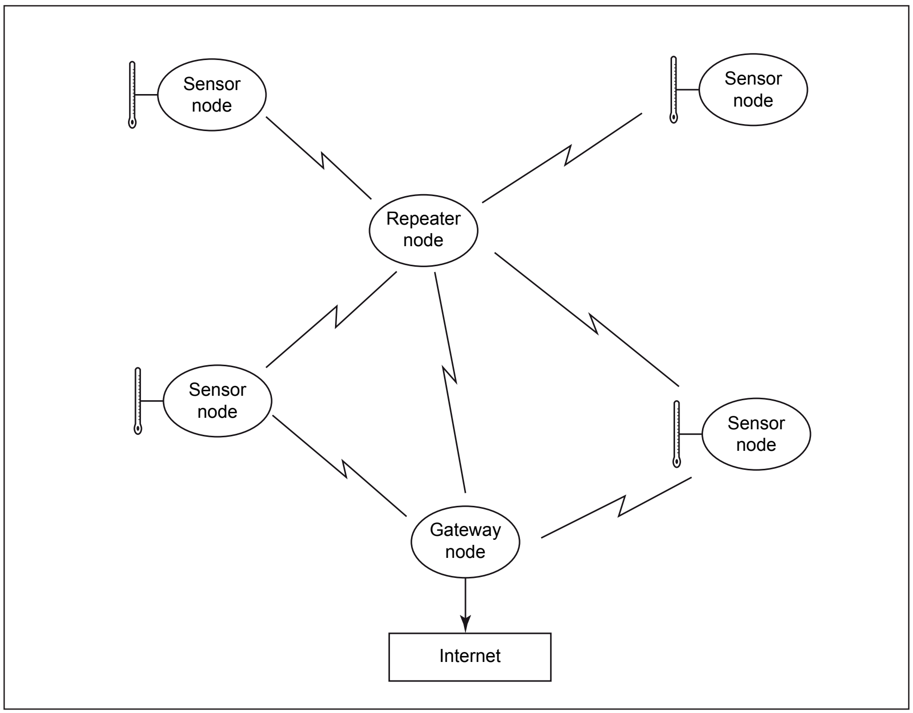


Fig. 1: UKHASnet mesh networking.

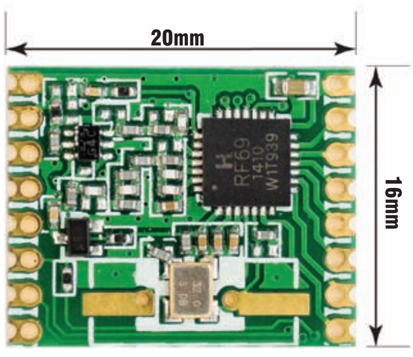


Fig. 2: The Hope RF RFM69HW VHF/UHF programmable transceiver.

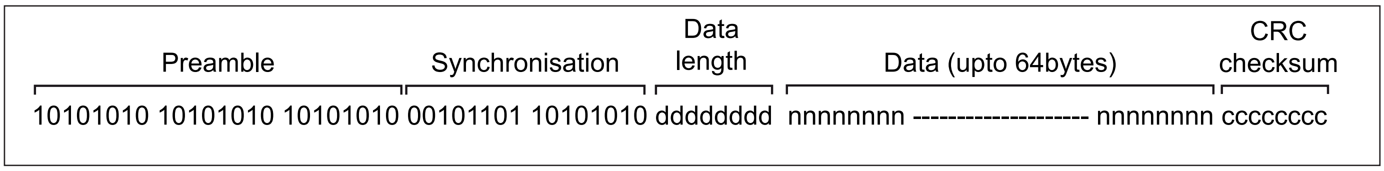

Preamble
Synchronisation
Data length
Data (upto 64bytes)
CRC checksum
10101010 10101010 10101010 00101101 10101010 dddddddd nnnnnnnn ------------------- nnnnnnnn cccccccc

Fig. 3 An illustration of an RFM69 packet.

check for damage. In the RFM69 modules the packets employ the following format, **Fig. 3**:

Preamble: This is a sequence of three bytes, each set to 0xAA. In binary notation this is 10101010, which some of you will recognise as the same technique used by RTTY operators as a tuning signal (in their case, RYRYRY). The receiver uses this alternating signal to adjust the automatic frequency control (AFC) and optimise the RF gain settings.

Synchronisation: This is a two-byte sequence of 0x2D and 0xAA that's used to synchronise the receiver. It's required so that the receiver can identify the beginning and end of each element in the message and trim its data clock to match the transmitter.

Data Length: A single byte that tells the receiver how many bytes of data to expect. In the present format this is limited to 64 bytes but this is more than adequate for most applications.

Data: This is where we put our message. If you need to send more than 64 bytes, you simply split the data over multiple packets.

Checksum: A two-byte checksum is added, which is used by the receiver to check that the data has been received error free.

The RFM69 series modules automatically handle the packet management so that all the user has to do is to supply the data to the transmitter and extract it from the receiver. Even this task has been made simple thanks to the availability of Open Source RFM69 libraries.

On the RF front, the UKHASnet employs simple 2-frequency FSK modulation at 2,000 baud with a shift of 24kHz. The operating frequency of 869.5MHz was chosen because it permits transmit powers up to 100mW from an airborne device in this licence-exempt band.

UKHASnet Data Format

If you consider UKHASnet as a layered protocol, then layer 1 is the RF modulation system, layer 2 is the packet format I've just described and layer 3 becomes the UKHASnet data format that fits inside the Data section of the RFM69 layer 2 packet, **Fig. 4**. When designing the UKHASnet data format, the first requirement was for it to be human readable. This makes the data very easy to use and also simplifies the debugging process when you have problems. Here are details of the packet format:

Repeat number: This is a single byte at the start of the message that specifies the number of times the message should be repeated. Every time a node repeats the message, this number is decremented by 1. A repeat limit is necessary to prevent repeater loops where the message would just go round and round between repeaters. If you want your node to permeate further through the network, you can increase this value for all originated packets.

Sequence Count: This is a single alphabetical character that increments from b to z and is used to show the order in which packets should be reassembled if you have long data messages to send.

Data: This is where the main content of the message is placed. Each item of data is preceded by a single letter to indicate the data type, with the values following as a comma-separated list. To send a series of voltage readings the data would look like this: V3.2,4.7,5.6 and so on.

Path: The final part of the message is enclosed in square brackets and gives the ID of the node originating the packet plus details of any nodes that repeated the packet. The node at the end of the list is the last one to transmit. Put simply, whenever a node repeats a packet, it reduces the repeat number by 1 and adds its ID to the end of the message.

I've also shown an illustration of the data format in **Fig. 5**.

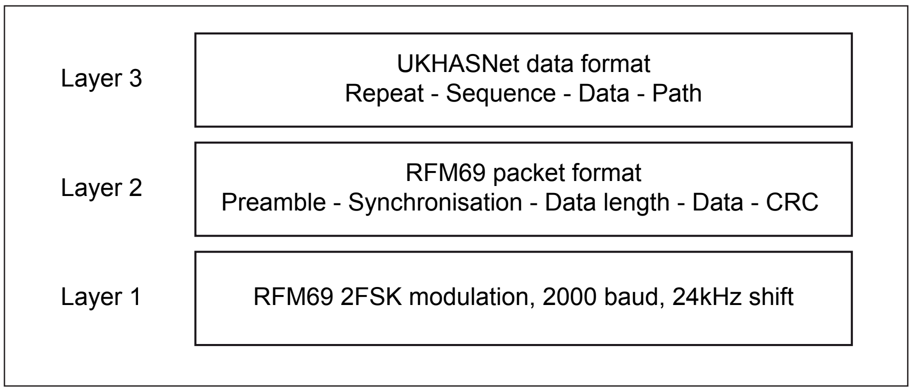

Layer 3
UKHASNet data format
Repeat - Sequence - Data - Path
Layer 2
RFM69 packet format
Preamble - Synchronisation - Data length - Data - CRC
Layer 1
RFM69 2FSK modulation, 2000 baud, 24kHz shift

Fig. 4: The layered protocol used for UKHASnet. Layer 3 sits inside the Data segment of Layer 2.

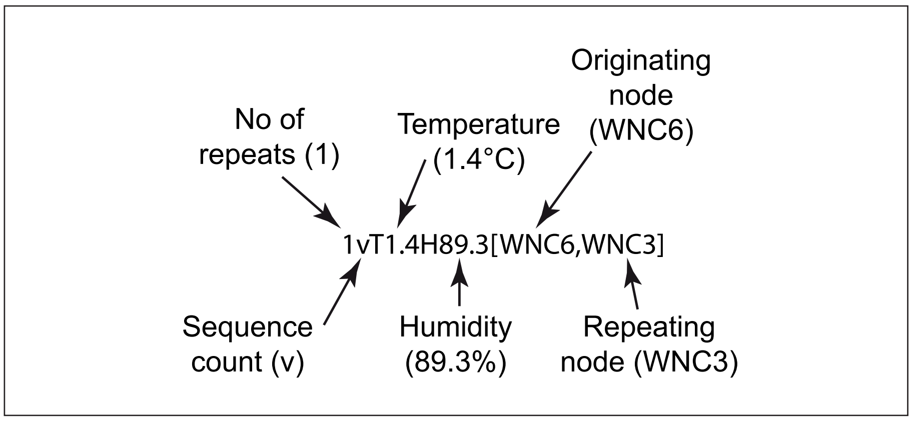


Fig. 5: The diagram shows a typical UKHASnet packet originating from one of my nodes, WNC3.

How to Use UKHASnet

The best way to learn how to use the network is to build yourself a couple of nodes so that you can start playing. To help with this process, there is an excellent WiKi on the UKHASnet website. This is undergoing continuous improvement and is a great reference source. There is also a very useful web API that can be used to monitor the data from your nodes. The web API not only gathers the data from your nodes but also displays the results, graphs the historical data and even shows activity on a map. The minimum you need to get started is a sensor node and a gateway node. If you have a Raspberry Pi kicking around the shack, this can be used to handle the internet linking to send your node data from the gateway to the web API. The code for this is already written so it's really a case of pulling the hardware together.

Stripboard Node

If you have PCB making facilities and are familiar with the Eagle schematic and PCB designer, you will find that there are designs and board layouts on the Github site to get you going. If you don't have those facilities, then the stripboard node option is a good place to start. You will find full details of the node at the URL below.
http://goo.gl/4n0ORm

All you need is an ATmega328 (with Arduino bootloader), RFM69HW 868MHz module, MCP1700 3.3V voltage regulator, 28-pin DIL IC Socket, 2 x 1μF ceramic capacitors, stripboard, DS18B20 sensor and a few resistors.

With so few connections, the stripboard node is quite straightforward to build. I've shown the schematic in **Fig. 6** and this has been laid out to align with the stripboard construction so it serves as a layout guide as well as a circuit diagram. You will see that I've added a DS18B20 temperature sensor to the circuit. This is only required on the sensor node. At the bottom of the stripboard section of the WiKi you will see a link to a new section I have supplied that deals with programming bare ATmega328s with a Windows PC.

Ready Built Nodes

If you're looking for a compact ready-built node, then the US-made Moteino by Low Power Labs is a good bet. The website link is at the end of this paragraph. These great boards measure just 33 x 23mm, comprise an ATmega328 with all the supporting circuitry and can be bought complete with an RFM69HW ready fitted for about £12 each, **Fig. 7**. The shipping costs are also very reasonable at just over £6 for USPS service to the UK. The ATmega328 on the Moteino board is supplied programmed with an Optiboot bootloader so it can be used with the Arduino IDE. As you can see from Fig. 6, the Moteinos have the RFM69HW fitted on the underside of the PCB, which helps to make the units so compact. Despite their small size, all the standard Arduino pins are brought out to the 0.1in-spaced headers. Due to the small size, there is no USB connection and the boards need to be programmed using a standard FTDI adapter lead. If you don't have one of these available, Low Power Labs can supply one for just under £10. Alternatively, you can use an ISP programmer or go for the MoteinoUSB with radio that costs around £16.
http://goo.gl/690RXd

Next month, I'll get into the practical details of building a network and adapting the software. If you want to get a head start, then the UKHASnet Wiki, is a great place to start and the IRC channel is a good place to visit if you have questions.
https://www.ukhas.net/wiki/doku.php

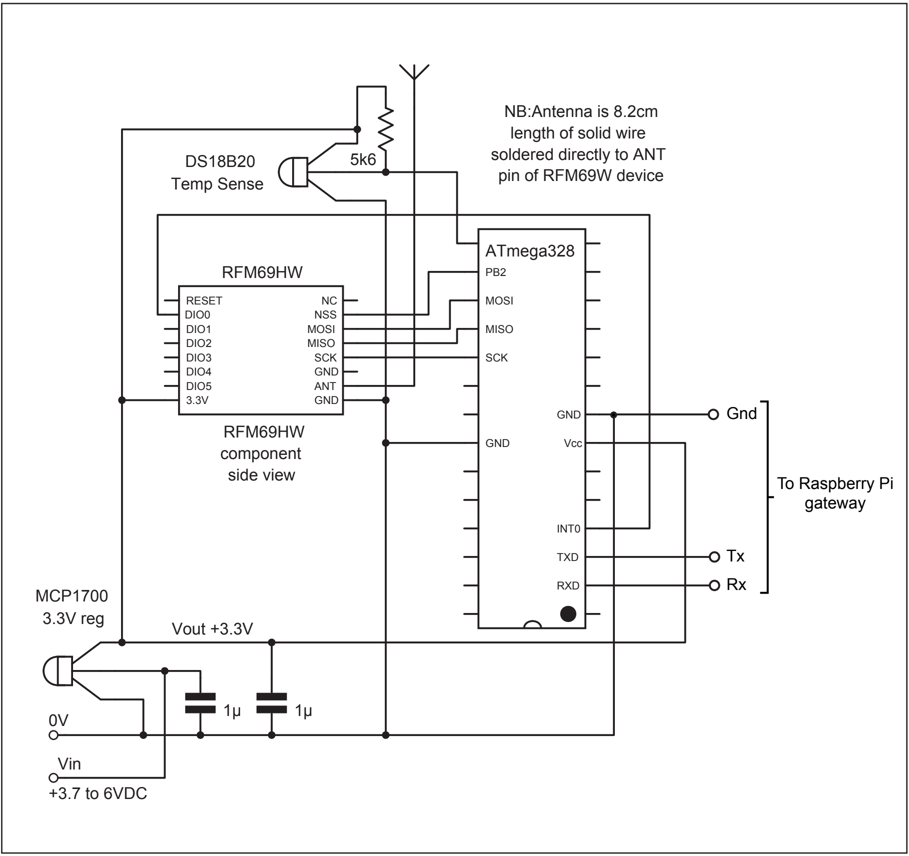


Fig. 6: Schematic and layout diagram of a stripboard node.

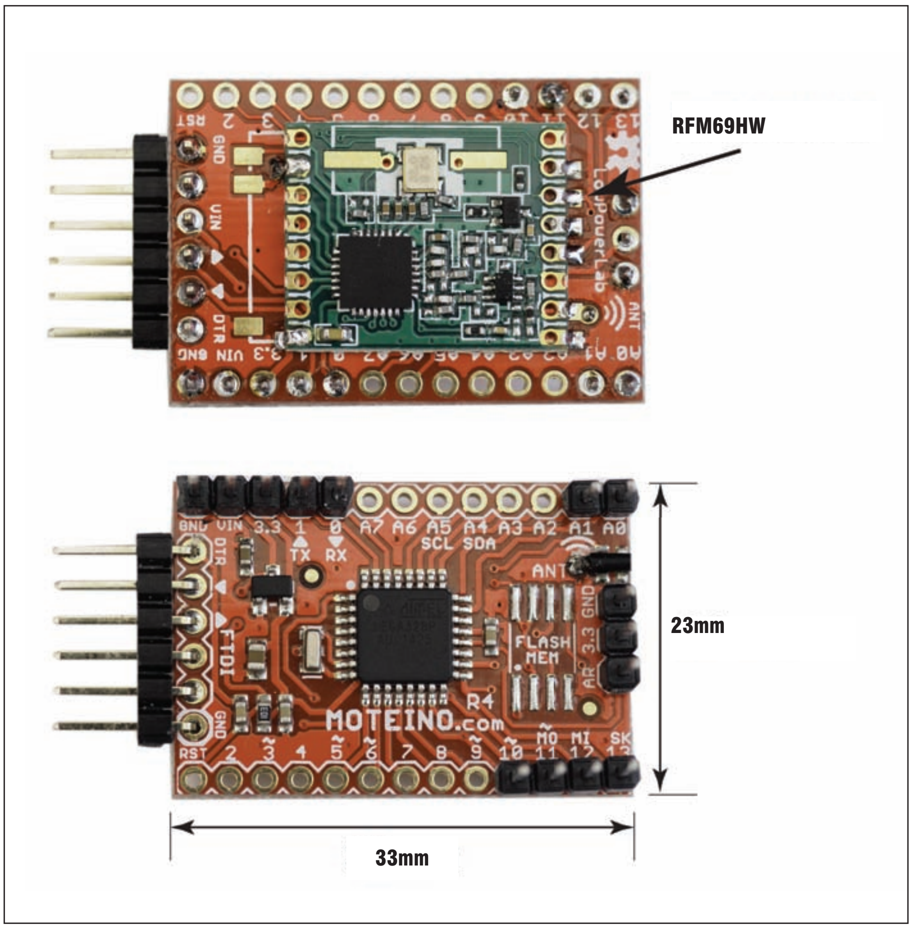
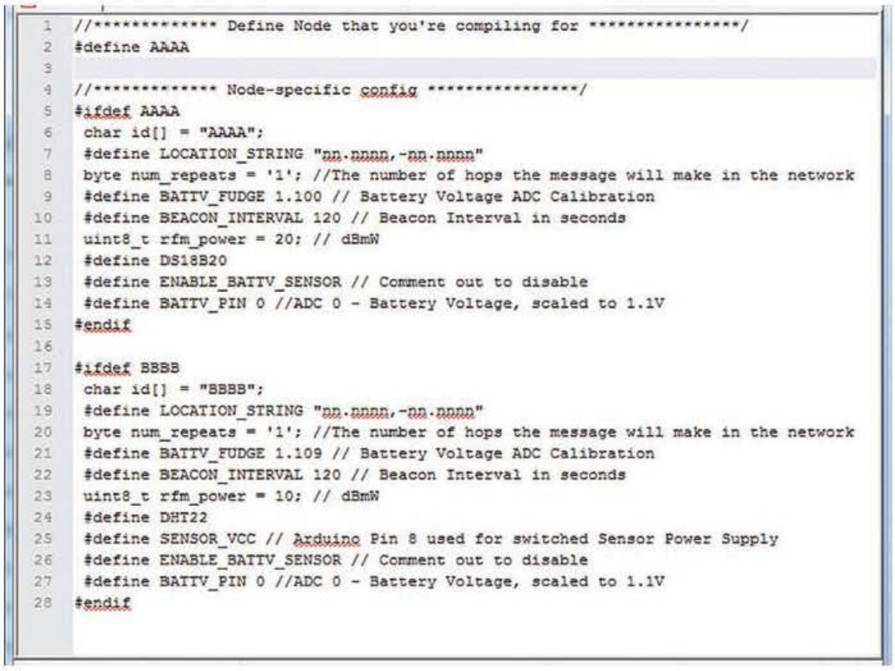


Fig. 7: The Moteino ready-built Arduino + RFM69 boards that are ideal for UKHASnet.



```
//************* Define Node that you're compiling for ***************/
#define AAAA

//************* Node-specific config ***************/
#ifdef AAAA
 char id[] = "AAAA";
 #define LOCATION_STRING "nn.nnnn,-nn.nnnn"
 byte num_repeats = '1'; //The number of hops the message will make in the network
 #define BATTV_FUDGE 1.100 // Battery Voltage ADC Calibration
 #define BEACON_INTERVAL 120 // Beacon Interval in seconds
 uint8_t rfm_power = 20; // dBmW
 #define DS18B20
 #define ENABLE_BATTV_SENSOR // Comment out to disable
 #define BATTV_PIN 0 //ADC 0 - Battery Voltage, scaled to 1.1V
#endif

#ifdef BBBB
 char id[] = "BBBB";
 #define LOCATION_STRING "nn.nnnn,-nn.nnnn"
 byte num_repeats = '1'; //The number of hops the message will make in the network
 #define BATTV_FUDGE 1.109 // Battery Voltage ADC Calibration
 #define BEACON_INTERVAL 120 // Beacon Interval in seconds
 uint8_t rfm_power = 10; // dBmW
 #define DHT22
 #define SENSOR_VCC // Arduino Pin 8 used for switched Sensor Power Supply
 #define ENABLE_BATTV_SENSOR // Comment out to disable
 #define BATTV_PIN 0 //ADC 0 - Battery Voltage, scaled to 1.1V
#endif
```

Fig. 6: An example NodeConfig.h file for UKHASnet.

Table 2

Parameter	Function
Char id[ ]	Unique id for your node.
LOCATION STRING	Lat and long in degrees and fractions.
Num_repeats	Sets the number of hops you want your node to make in the network.
BATTV_FUDGE	This is a correction factor used to compensate for inaccuracies in the ATMega328s ADC. Start at 1.1V.
BEACON_INTERVAL	Sets the interval (seconds) between beacon transmissions.
Unit8_t rfm_power	This sets the output power of the RFM69 in dBW.
DS18B20	Include this if you're using the DS18B20 temperature sensor.
DHT22	Include this if you're using the DHT22 humidity and temperature sensor.
SENSOR_VCC	Include this if you're using a DHT22 sensor and want its power supply switched to conserve battery power.
ENABLE_BATTV_SENSOR	Enables the battery voltage measurement.
BATTV_PIN	This selects ADC 0 to measure the battery voltage.

UKHASnet

There's just a bit of space left to continue my look at UKHASnet and give you a few pointers on how to use the software. The nodes that I described last month are all based around the popular ATMega328 microcontroller that forms the basis of the popular Arduino boards. Although the UKHASnet nodes are not actually Arduino boards, they can be programmed using the Arduino Integrated Development Environment (IDE). Not only is the Arduino IDE easy to use but there is also a huge range of prewritten libraries available. These libraries can be used to construct quite advanced projects with a minimal amount of code to link the library functions together. The software for the ATMega328-based UKHASnet nodes is prewritten and available for free download from their github repository at the URL below.
http://goo.gl/JFKMro

One of the unusual features of this software is that the libraries necessary for the project are included in the download folder. I mention this because if you already have any of those libraries installed in your Arduino IDE, you may see errors when you come to program the chips. This is because the Arduino IDE uses its pre-installed libraries before using those in the project directory. The main culprit is the RFM69 library because the UKHASnet version has been modified. At some point, the team will probably tidy this up by changing the library name but in the meantime it's as well to be aware. I've written detailed step-by-step instructions for programming ATMega328 nodes using Windows in the UKHASnet WiKi and you can see that at the URL below.
http://goo.gl/uVE0gE

Configuring Nodes

Before you program your nodes, you need to allocate a unique ID to each node and set a few other parameters. This process has been made quite simple through the provision of a NodeConfig.h file. This file is used to set the main variables for each node and you can use one file to hold the settings for all your sensor or repeater nodes. This is achieved by using the #ifdef pre-processor command. I've shown an example NodeConfig file in **Fig. 6**. Here you can see that at the top of the file, you change the #define line to show the node you want to program. When the IDE compiles the software, it will check the #define line and include the configuration settings for only that node. I've shown the purpose of the configuration settings in **Table 2**. Note that you will find an example configuration file called NodeConfig-Template.h but you need to remember to rename it to NodeConfig.h before you start compiling.

For more information on using UKHASnet nodes take a look at their WiKi, which is regularly updated. Their #UKHASNet IRC channel is also a great source of help.