REDRAT

The irNetBox Network Control Protocol Mk-I, Mk-II and MK-III

Chris Dodge – RedRat Ltd

08 May 2015

V3.03

1 Introduction

The irNetBox is a networked device for infrared remote control of multiple devices, so allowing computer control of audio-visual equipment placed within proximity of a network access point.

For development of applications on the Windows platform (.NET or COM), the RedRat SDK can be used which hides the details of all network data transfer and presents a simple API. If however you are developing an application for a different platform or prefer to use direct network IO, then this document describes the details of the network control protocol.

The protocols described here apply to both three generations of irNetBox, which are termed the MK-I, MK-II and MK-III (or I, II and III for short). Each generation of irNetBox introduces new functionality so some care must be taken to ensure that the correct messages are sent to the right type of hardware, though we have attempted to make them as compatible as possible.

2 The irNetBox Design

2.1 Overview

2.1.1 irNetBox-I

- One IR generator microcontroller (IR signals can be routed to any output port combination)
- 16 IR output ports
- Optional high-power output on port 1
- IR input for learning IR signals from original handsets
- Network and USB interface

2.1.2 irNetBox-II

- RoHS compliant (lead-free).
- All IR output ports have three power levels. On the original irNetBox, only output 1 has switchable high and low power.
- Faster IR micro providing support for a wider range of IR signal types.

2.1.3 irNetBox-III

- Uses the powerful four core XS1-G4 from XMOS as its main processor
- Can generate 16 different IR signals concurrently
- IR outputs have 100 different output power levels
- Built in self-test hardware
- Additional 25-way D-type socket on rear for connecting IR emitter bundles

2.2 Hardware Architecture

As the network control protocol is fairly low-level, some knowledge of the hardware is useful to understand the type and sequence of commands needed for operation. Figure 1 shows the main elements of the MK-I and MK-II hardware while Figure 2 shows the MK-III architecture.

2.2.1 XPort Network Interface

Developed by Lantronix (www.lantronix.com), this is a complete network interface in a single unit. Its main role is to convert TCP/IP data packets into serial data to pass to the IR I/O micro, and visa-versa. See section 3 for details on network configuration of the XPort.

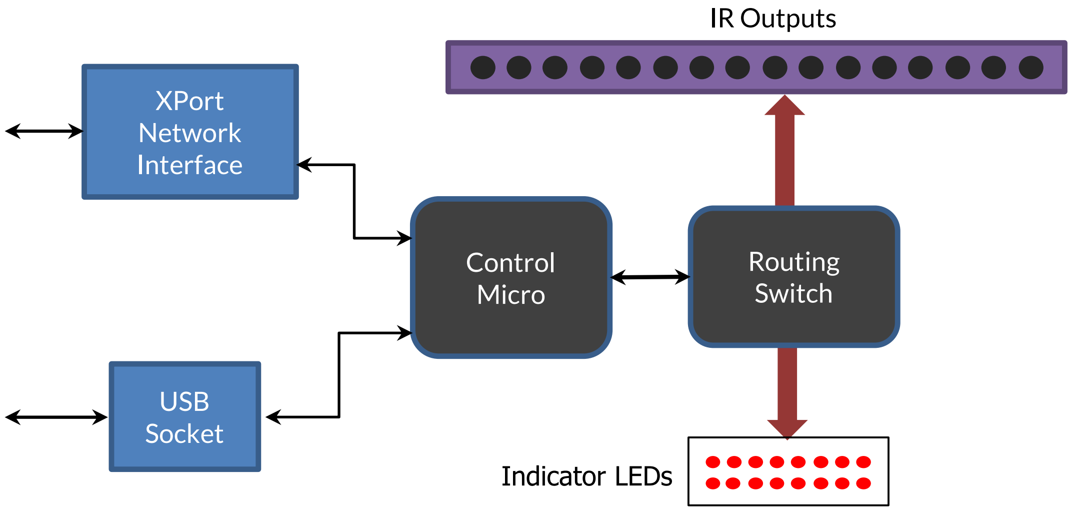


Figure 1. Overview of irNetBox-I and II architecture

2.2.2 Control/USB Micro

This micro is responsible for IR signal sampling and signal generation in the MK-I and MK-II units. It also has a parallel interface to the signal routing logic device, so generates the commands to configure the routing device for a particular operation, for example, routing the generated IR signal to the requested IR outputs.

In the MK-III system the same microcontroller is used to handle USB comms and forward on to the XS1 processor.

2.2.3 Routing Switch (I & II only)

This is a complex programmable logic device (CPLD) that routes input and output signals to the correct destinations. It is instructed which outputs to enable by the control micro, and indicates its current status using the visible IR LEDs. It is sufficiently flexible to allow any combination of IR outputs to be selected at whichever power level is required.

2.2.4 XMOS XS1-G4 Processor (III only)

This is a 32-bit, four core processor with a multi-threaded processor architecture and so capable of generating up to 16 different IR signals concurrently.

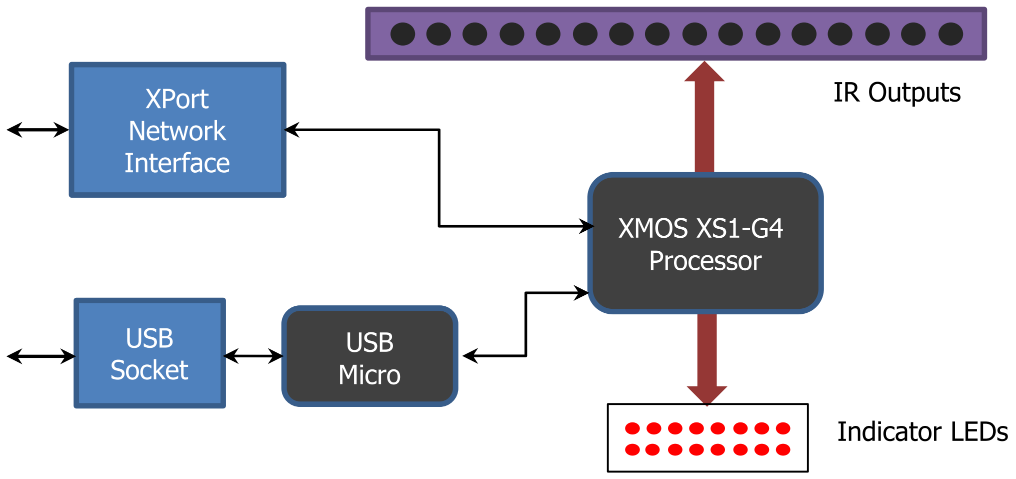


Figure 2. Overview of the irNetBox-III architecture

3 Network Installation and XPort Configuration

The irNetBox has to be setup to work on your network, and Lantronix provide a tool for the configuration of the XPort called the *Device Installer*. It can be downloaded from the Lantronix website at http://www.lantronix.com/ and can be used to assign the device an IP address or allow it to use a DHCP server on the network for obtaining an address.

The Lantronix Device Installer should not be used to adjust other XPort parameters as this may prevent correct operation within the irNetBox. In case the XPort configuration needs to be re-set for correct operation, then please use the Windows *IRNetBox Manager* application, which can be downloaded from the RedRat website: http://www.redrat.co.uk/.

4 IRNetBox Discovery and Identification

4.1 Discovery

Once the irNetBox has been installed on a network, your application code may need to discover the IP address of any irNetBoxes on the local network. If you have assigned a fixed IP address to the device, this step is not necessary, but if the IP address is obtained via DHCP or devices are frequently taken in and out of commission, then dynamic discovery is needed.

Discovery is done with a UDP broadcast, sending a block of 4 bytes as shown below to port 30718 (0x77FE).

UDP Broadcast for irNetBox Discovery	
To Port:	30718 (0x77FE)
Data block to send	0x00, 0x00, 0x00, 0xF6

The XPort will respond with a datagram of 30 bytes in length, containing the following information in the order given:

Datagram Returned from the 0xF6 Request		
Data block type	4 bytes	Should be 0x00, 0x00, 0x00, 0xF7
First few bytes of firmware image ID	16 bytes	Bytes 4 and 5 of this section contain the firmware type – “X1”, “X2”, “X5” or “X9”.
Device info.	4 bytes	
MAC address	6 bytes	The MAC address of the XPort

The MAC address is unique for every device on a network, so can be used as a method of uniquely identifying the irNetBox.

UDP communication information should also make it possible to obtain the source IP address of all returned datagrams, so if the IP address is not known, it can be obtained this way.

4.2 Identifying irNetBox Type

As there are some differences between the MK-I, II and III irNetBoxes, software systems need a method of identifying the kind of irNetBox with which they have a connection.

4.2.1 Identification via UDP

The first step of this process is to identify the XPort firmware ID as given in the previous section:

XPort Type	irNetBox Type
X1 or X2	MK-I
X5	MK-I, MK-II or MK-III
X9	MK-III

If the XPort type is an X1 or X2, then the irNetBox is a MK-I version and no other identification communication is needed.

The X5 XPort types have been used in both MK-I, II and II irNetBoxes, so further interrogation of the XPort is needed as it contains an identifying label. To obtain the block of configuration information containing the label, a UDP request is sent to port 30718 (0x77FE), and a 130 byte data block is returned.

UDP Request for irNetBox Type Discovery	
To Port:	30718 (0x77FE)
Data block to send	0x00, 0x00, 0x00, 0xE4

The following 130 byte data block is then returned:

Data block Returned from the 0xE4 Request		
Misc data	32 bytes	Not of interest
Length of irNetBox type label	1 byte	Length of label = `dataBlock[32]`
Label	X bytes	The label, either *REDRAT4*, *REDRAT4-II* or REDRAT4-III.

A recommended method of checking the returned data is firstly to check `dataBlock[32]` to see if it is greater than 0. If it is 0 when using an X5 firmware type XPort, then the device is not an irNetBox. If it is greater than 0, then read the label from `dataBlock[33]` onwards as a null terminated string, i.e. until a byte of value zero is encountered.

The X9 firmware type of XPort is used in MK-III irNetBoxes only from September 2013 onwards, and serial number 15892 or higher. With this type it is recommended that the above procedure is followed for the identification of the irNetBox type to support future versions.

4.2.2 Identification via Direct TCP Communication

Once a connection with the irNetBox has been established, then using message 9 will give the irNetBox's version. See section 5.2.6 for details of the data returned and using this to identify the specific hardware version.

5 IR I/O Comms

This is the method by which the irNetBox can be instructed to output and capture remote control signals. The XPort passes through any data sent via a TCP/IP socket opened on port 10001 to the IR I/O micro and all data returned from the control micro is then passed back via the same TCP/IP socket to the host. By keeping this socket open, the host can ensure that it has exclusive access to the irNetBox for the duration of its session.

5.1 Control Protocol Overview

For each message that is sent to the irNetBox, an acknowledgement message will be returned, so the host should not send a second message until the acknowledgement from the first message has been received.

The structure of a message from host to irNetBox is:

Message Structure: Host to irNetBox		
'#'	byte	The '#' character indicates to the control micro the start of a message.
Message length	ushort	The length of the data section of this message.
Message type	byte	One of the values listed below.
Data	byte[]	Any data associated with this type of message.

A **ushort** value is a 16-bit unsigned integer in big-endian format. Most PCs are little endian, so the bytes need to be swapped on the host.

Once the control micro has received a message, it will respond with a very similar type of message, except that it does not have the '#' start of message character:

Message Structure: irNetBox to Host		
Message length	ushort	The length of the data section of this message.
Message type	byte	Contains either: a) The same value as the original message from the host or b) A value indicating "Error".
Message Data	byte[]	Any data associated with this type of message.

5.2 Message Types

The table below lists the following message types:

Message Types For Commands to and from the irNetBox.		
Decimal	**Hex**	**Description**
1	0x01	Message contains error information.
4	0x04	Read the IR I/O micro firmware ID.
5	0x05	Turn power to the CPLD device on.
6	0x06	Turn power to the CPLD device off.
7	0x07	Send an instruction to the CPLD device.
8	0x08	Read the serial number of the device.
9	0x09	Read the device version.
Modulated IR Signal Functions		
16	0x10	Allocate memory for a modulated IR signal on the IR I/O micro.
17	0x11	Download modulated signal data to the IR I/O micro memory.
18	0x12	Output the modulated signal stored in memory.
19	0x13	Initiate learning of a modulated IR signal
20	0x14	Cancel signal learning.
21	0x15	Signal data upload, i.e. *irNetBox* to host.
IrDA-Like Signal Functions		
22	0x16	Allocate memory for IrDA data packets.
23	0x17	Download an IrDA packet for output.
24	0x18	Output the IrDA packet in memory.
26	0x1A	Output a flash-code IR signal (no carrier wave).
Signal Capture and Memory Parameters		
32	0x20	Read the signal capture and memory parameters.
33	0x21	Sets the signal capture and memory parameters.
IR Output Bit Mask (II and III only)		
34	0x22	Set all outputs at the correct power levels in one instruction, downloaded as a bit mask.
Asynchronous IR Output Commands (III only)		
48	0x30	Output IR signal asynchronously
49	0x31	Asynchronous IR output is complete

To support the high-throughput of IR signals in the MK-III irNetBox, the asynchronous/overlapped messages 0x30 and 0x31 have been introduced. These are explained in detail in section 6.

Due to the difference in architecture between the MK-I,II irNetBoxes and the MK-II box, not all commands listed below make sense for the MK-III. However it attempts to be as backwards compatible as possible, so will attempt to make sense of the commands in such a way as to result in identical IR output behaviour.

5.2.1 [Message 1] Error

The first byte of the message data contains the error code, the values it can currently take being listed in the table below.

Error Values Returned From the irNetBox		
Decimal	**Hex**	**Description**
32	0x20	Signal Capture: The initial IR signal pulse is not long enough to measure the carrier frequency. It usually expects to eight. To correct, try reducing the number of carrier cycles to count, see section 5.2.16.
33	0x21	Signal Capture: Not enough "length" values allocated for this IR signal, indicating that the environment is noisy or the signal is complex. To correct, increase the number of lengths, see section 5.2.16.
34	0x22	Signal Capture: Not enough memory has been allocated for the sampled signal data. The amount of memory can be increased, see section 5.2.16.
35	0x23	Signal Capture: Too many repeats in signal, i.e. it exceeds the maximum allowed, which is 255.
40	0x28	Signal Download: Not enough memory on device to allocate memory for modulated signal. To correct this problem, reduce some of the memory data size parameters – see section 5.2.16.
41	0x29	Signal Download: No memory for modulated signal has been allocated. To correct, request signal memory allocation – see section 5.2.7.
42	0x2A	No signal data has been captured or downloaded. This happens if an attempt is made to output an IR signal before any signal data has been downloaded.
43	0x2B	Signal Download: Not enough memory available on device for allocation of the IrDa signal buffer.
44	0x2C	Signal Download: No memory for the IrDa data has not been allocated. To correct, send command to allocate memory for an IrDA signal download.
45	0x2D	Signal Download: No IrDa signal data has been downloaded. This happens if an attempt is made to output an IrDA type signal before the data has been downloaded.

5.2.2 [Message 4] Read Firmware Version

The data section of the returned message will contain a null terminated ASCII string, such as:

```
1.00IP (c) RedRat Ltd 2008
```

5.2.3 [Message 5 and 6] CPLD Power Management

The CPLD can have its power turned off under software control so that when idle, the irNetBox does not draw so much power, saving up to 20%. Additionally all power to the red and infrared LEDs is fed through the same power switch.

Turning the CPLD power on takes about 50ms (up to 250ms on the irNetBox-I), so turning the CPLD power off is not necessarily practical following the output of every IR signal. A reasonable rule of thumb is to enable the CPLD power once a TCP/IP connection has been established with the irNetBox, and disable the power on disconnect.

The CPLD power is off when the irNetBox is first started.

5.2.4 [Message 7] CPLD Instruction

The first byte of the message data contains the instruction for the CPLD device. The IR outputs are numbered 1 to 16 below, corresponding to the labelling on the irNetBox. If you have used the SDK, then note that the numbering starts from 0.

CPLD Instructions		
Decimal	**Hex**	**Description**
0	0x00	Reset the CPLD. The reset state is where none of the red and infrared outputs are enabled, and IR input is not enabled.
2	0x02	Enable <u>low power</u> on the first IR output, i.e. number 1.
3 to 17	0x03 to 0x11	Enable <u>low power</u> IR outputs 2 to 16.
18	0x12	Enable all IR outputs.
19	0x13	Enable high power on output 1.
20	0x14	Route the input from the "learning" remote control receiver to the IR I/O micro. It should not be necessary to explicitly send this instruction as the firmware does this automatically.
23	0x17	Instructs the CPLD to set the visible LEDs to reflect the enabled state of the IR outputs, for example, if IR output 1 is enabled, then the red LED 1 will light.
24	0x18	Visible LEDs do not reflect whether the IR outputs are enabled or not, i.e. they remain off.
The following instructions are for the irNetBox-II only		
32	0x20	Enable <u>medium power</u> on output 1.
33 to 47	0x21 to 0x2F	Enable <u>medium power</u> on outputs 2 to 16.

To generate <u>high power</u> on any output, enable both the <u>low</u> and <u>medium</u> power, which together will generate the high power signal (irNetBox-II only).

Rather than sending a large number of single instructions to set all outputs to the required state, a single bitmask can be used – see section 5.2.18 (irNetBox-II and III only).

Even though the irNetBox-III does not have a CPLD device, it will interpret these commands to perform a the equivalent action.

5.2.5 [Message 8] Read the Device's Serial Number

The serial number is a 4-byte number, for example '00-00-2E-27' in hex.

When reading this value, the data structure returned is based on a USB descriptor which is 18 bytes in length in this case:

Data structure of the serial number descriptor		
Descriptor length	byte	Should always be 0x12 (18).
Descriptor type	byte	Should be 0x03
Serial number	ushort[8]	A unicode string containing 8 characters of the hex serial number.

This returns a 4 byte serial number that has been programmed into the IR I/O micro. Although the MAC address of the TCP/IP version of the *irNetBox* can be used as unique id, future USB and serial port based versions will use this serial number.

For the serial number '00-00-2E-27', the byte data returned would be:

0x30, 0x00, 0x30, 0x00, 0x30, 0x00, 0x30, 0x00,

0x32, 0x00, 0x45, 0x00, 0x32, 0x00, 0x37, 0x00

5.2.6 [Message 9] Read the Device Version

When reading the device version, a descriptor containing many device attributes is returned. The actual payload data is an 18 byte array, the following table giving the structure and listing the bytes of interest:

Data structure of the device descriptor			
Descriptor length	byte	Should always be 0x12 (18).	
Descriptor type	byte	Should be 0x01	
Payload data	byte[16]		
		byte[0] – byte[5]	
		byte[6] – byte[7]	USB vendor ID for RedRat Ltd – LSB first
		byte[8] – byte[9]	Product ID – LSB first
		byte[10] – byte[11]	Product version ID – LSB first
		byte[12] – byte[15]	

This is a 2 byte number (major, minor version) that reflect the hardware version of that particular product, usually reflecting PCB modifications.

The primary data of interest is the product ID, which for irNetBoxes are:

irNetBox Product IDs	
MK-I	PID = 2
MK-II	PID = 7
MK-III	PID = 8

5.2.7 [Message 16] Allocate Memory for a Modulated Signal

This tells the IR I/O micro to allocate memory for modulated IR signal data before a modulated signal is downloaded for output.

The reason that this memory allocation is not automatic is that data transfer between the XPort and the IR I/O micro has no flow-control, and there is not sufficient time to dynamically allocate memory during the signal download operation without losing some data.

Once memory has been allocated for this type of IR signal, it does not need to be de-allocated. Also, allocation can be called multiple times without problem, the firmware tracks whether it is necessary to actually execute the memory allocation, so it can be called before every signal download.

Note: This message must be called before the first modulated signal is downloaded or the following the output of an IrDA-like IR signal.

5.2.8 [Message 17] Download Modulated IR Signal Data

Once memory has been allocated for a modulated IR signal, the signal data can be downloaded. The data structure for a modulated signal is shown in the table below, with descriptions of the fields following that. Please also refer to section 8 for further information on infrared remote control signals.

Data structure of a Modulated IR Signal	
Information Structure	
Intra-signal Pause	uint
Modulation frequency timer count.	ushort
No. of periods over which mod. freq. is measured	ushort
Maximum number of lengths allowed (alloc'd)	byte
Actual number of length values	byte

Maximum allowed signal data size (bytes)	ushort
Actual size of signal data	ushort
Number of signal repeats	byte
Data Arrays	
Length data array.	ushort[maxLengths]
Signal data array	byte[]

Important: Before downloading a signal, the values that the *irNetBox* has for the size of the length array (maximum number of lengths) and the signal data must be set to the same value as that given in the signal data structure. This allows the downloaded data to be placed directly over the signal memory "footprint" in the *irNetBox*. If the downloaded signal data block does not fit onto the memory footprint, then outputting the signal will not work.

How is this best achieved? The simplest method is to always set the signal memory parameters to that of the signal before every download – see Messages 32 and 33. On a network with a high latency or in the situation where IR signals are being output in rapid succession, the small delay imposed by this operation may not be acceptable. In this case, a local copy of the length and signal data array sizes can be held and kept in sync with the *irNetBox*, and used for checking against the signal to be downloaded. In the vast majority of cases, the *irNetBox* and signal data will both have the default values.

The RedRat signal database utility can be used to generate signal data blocks from XML signal databases for reading into an application to download – see section 10 for further details.

5.2.8.1 Intra-Signal Pause

Period of dead time between the main and the repeat signal sections. This value is that of a counter running at 2MHz, so the length can be calculated as:

$$counter\_value = \frac{pause\_length\_in\_mS}{1000} * 2MHz$$

As with all downloaded ushort or uint values, it has to be big-endian.

5.2.8.2 Modulation Frequency Timer Count

The timer producing the modulation/carrier frequency switching runs at 6MHz, so converting a required frequency value in Hz (e.g. 38000Hz), the following equation is used:

$$timer\_value = 65536 - \frac{6MHz}{carrier\_freq\_in\_Hz}$$

As the *timer_value* is an up-counting re-load value, we subtract it from 65536 to give the correct timer count to overflow.

Note: The value contained in this field is slightly different between signal download (this section) and signal upload when capturing/learning IR signals (section 5.2.12). The reason for this is that the way the carrier frequency is measured (for upload) is different from the way in which the carrier frequency is generated (for signal output).

5.2.8.3 No. of Periods over Which the Modulation Frequency is Measured

For signal download, this value can be set to zero. It is used on signal input to calculate the carrier frequency.

5.2.8.4 Maximum Number of Lengths Allocated

As can be seen from section 9.1.2, length values are the "alphabet" that can be used to construct a signal. An average IR signal has less than 8 length values, however by default the irNetBox allocates enough memory for 16 values.

Why is this value needed? This value is used on signal **upload** (i.e. transferring an IR signal from the *irNetBox* to the PC) only so that the application knows the length of this array in the data block. The array of length values returned as part of the signal is *maxLengths* in size, even if they are not all used in the signal. On signal **download**, this value is not used as the memory allocated to hold this data must be configured before sending the IR signal data – see messages 32 and 33 for details.

5.2.8.5 Actual Number of Lengths Used

The number of elements in the length data array that contain values that are used in the signal array.

5.2.8.6 Maximum Allowed Signal Data Size

This is the size of data allocated for the signal data array. The "signal data" in this context is the array of values of the *length* alphabet, e.g. 1233332323… See section 9.1.2 for further details.

This value is used on signal **upload** (i.e. transferring an IR signal from the *irNetBox* to the PC) only so that the application knows the length of this array in the data block. The array of length values returned as part of the signal is *maxLengths* in size, even if they are not all used in the signal. On signal **download**, this value is not used as the memory allocated to hold this data must be configured before sending the IR signal data – see messages 32 and 33 for details.

5.2.8.7 Actual Size of Signal Data

The actual length of the signal data in the returned signal data array. The signal data array follows the length data array, so the start of this array is at:

```
    sizeof(information_structure) + (maxLengths*sizeof(ushort))
```

The signal data is contains both the *main* and *repeat* signal data sections (see section 9.1.1), in the following form:

```
    main_signal, EOS, repeat_signal, EOS
```

where the **EOS** (end of signal) marker is a single byte of value 127 (0x7F). If there is no *repeat* signal data then the two EOS markers will follow one another.

5.2.8.8 Number of Signal Repeats

When a remote control button is held down for anything more than a short press, the *repeat* is usually sent multiple times – this value holding the actual number of times. When downloading a signal, this value can be varied depending on whether a short or long button press wants to be reproduced.

5.2.8.9 Length Data Array

This is the signal "alphabet" (or look-up table) that holds the length of time values that are used to construct the signal. Each value represents a period in ms, but converted to the number of counts of a 2MHz timer, so the value is given by:

$$length\_value = \frac{length\_in\_mS}{1000} * 2MHz$$

The values in the length array have to be big-endian for download to the *irNetBox*, so may need to be converted.

5.2.8.10 Signal Data Array

The actual signal data as a byte array, for example "01223232323…". Each value is a lookup into the length data array.

5.2.9 [Message 18] Output Modulated IR Signal

Following signal download, the IR signal can be output, and so will be transmitted though all enabled IR outputs.

5.2.10 [Message 19] Initiate Signal Learning

This message prepares the IR I/O micro for signal input from the "learning" IR detector. This preparation includes: Allocating signal memory if necessary and setting the CPLD device to route input from the learning IR detector through to the micro. As with all other messages given so far, once the micro has finished preparation for signal input it will send an ACK message back.

Following this, the host has to wait for a further input message once the remote control button has been pressed, which will contain the IR signal data (i.e. message 21). This is the only situation in which a message is to be expected from the irNetBox that is not sent as a direct response from a message from the host.

The signal in waiting state can only be exited through:

- A signal being received and sent to the host or
- A signal input cancel message being sent.

5.2.11 [Message 20] Cancel Signal Input

This will disable signal input, unless signal sampling has already started. The returned message will therefore be either the signal input cancel ack. or input signal data.

5.2.12 [Message 21] Signal Data Upload

When the *irNetBox* returns the IR signal data during a learning operation, it is done with this message type.

The modulated IR signal data uploaded is **ALMOST** of the same structure as that described for download (in section 5.2.8.) however there is one difference, which is the value for the carrier/modulation frequency. This means that the uploaded data block cannot be directly downloaded for output without changing this one value. The reason for this is that the way the carrier frequency is measured (for upload) is different from the way in which the carrier frequency is generated (for signal output).

In the uploaded signal data, the carrier/modulation frequency is calculated with the following equation:

$$carrier\_freq\_in\_Hz = \frac{2MHz * No\_of\_periods\_measured}{carrier\_frequency\_count\_value}$$

where the *carrier_frequency_count_value* is the value from the field called "Modulation frequency timer count" in the table in section 5.2.8 – a big-endian ushort value.

The *No_of_periods_measured* value is the "No. of periods over which mod. freq. is measured" entry in the table given in section 5.2.8.

5.2.13 [Message 22] Allocate memory for IrDA data packets.

Instructs the IR I/O micro to allocate memory for an IrDA-like signal before it is downloaded for output.

The reason that this memory allocation is not automatic is that data transfer between the XPort and the IR I/O micro has no flow-control, and there is not sufficient time to dynamically allocate memory during the signal download operation without losing some data.

Once memory has been allocated for this type of IR signal, it does not need to be de-allocated. Also, allocation can be called multiple times without problem, the firmware tracks whether it is necessary to actually execute the memory allocation, so it can be called before every signal download.

Note: This message must be called before the first IrDA-like IR signal is downloaded or the following the download/output of a standard modulated IR signal.

5.2.14 [Message 23] Download an IrDA Packet Data

Following memory allocation for an IrDA-like signal, it can be downloaded using this message. Although it is easier to use the binary data converted directly by the Signal Database Utility for download (see section 11), the download data structure is also given here.

An IrDA-like IR signal is made up from a set of subpackets with a pause between each one. Each subpacket itself is made from a set of short pulses with varying time between the pulses, and in this fashion the data to be transmitted is encoded. See section 9.2 for more detail.

Data structure of an IrDA-like Signal for Download	
Information Structure	
Number of subpackets	ushort
Inter-subpacket pause length	ushort
Data Arrays	
Signal data array	byte[]

Important: Before downloading a signal, the size of memory allocated for the IrDA signal data storage should be set to be at least as large as the total signal data (data array + information structure size). This is set with the *Size of Memory* parameter given in section 5.2.16.2.

5.2.14.1 Number of Subpackets

This is a big-endian 2-byte number giving the number of subpackets contained in the data array.

5.2.14.2 Inter-Subpacket Pause Length

The time between the end of one subpacket and the start of the next. Given a value in ms (typically around 20ms), the value to download is given by:

$$download\_value = 65536 - \frac{pause\_length\_in\_mS}{1000} * 2MHz$$

The download value is big-endian.

5.2.14.3 Signal Data Array

Each subpacket is represented by a byte array, and these are concatenated to form the full data array.

Subpacket Data Array	
Number of elements	byte
Subpacket data	byte[]

The *Number of Elements* is just the length of the subpacket data for this subpacket, and each byte in the array is given by:

$$download\_value = 256 - \frac{pulse\_gap\_in\_mS}{1000} * 2MHz$$

5.2.15 [Message 24] Output IrDA Packet

Following the download of an IrDA signal, this message will output the signal via any enabled IR outputs. This message may be called any number of times to output the IrDA signal in memory.

5.2.16 [Message 32] Read the Signal Capture/Memory Parameters

The signal sampling algorithm can be adjusted using a number of parameters, the current values of which can be read as a single data block using this message.

Data structure of the signal capture parameters data block		
	Type	**Default Value**

Maximum number of lengths	byte	16 values
Size of memory allocated for the signal data array	ushort	512 bytes
Periods of carrier wave to measure	byte	8 periods
Length measurement "fuzz"	byte	112
Pause measurement timeout	uint	300,000
Minimum pause value	byte	115

5.2.16.1 Maximum Number Of Lengths

The maximum number of length values (the signal alphabet) that a can be used in a single signal. As memory on the irNetBox is limited, there is a trade-off in memory usage between the signal data and the length array. The default value is 16 lengths, however for complex signals it can sometimes be necessary to increase this value when learning a signal.

If a signal for download has a length array size different from the value set in the *irNetBox*, then the signal output will not work. It is therefore necessary to check this before each signal download or keep a copy of this value on the host, ensuring that it is kept in sync with any changes on the *irNetBox*.

5.2.16.2 Size of Memory for the Signal Data Array

The amount of memory used to hold the actual signal data. Default value is 512 bytes.

If a signal for download has a data array size larger than the value set in the irNetBox, then the signal output will not work. It is therefore necessary to check this before each signal download or keep a copy of this value on the host, ensuring that it is kept in sync with any changes on the irNetBox.

5.2.16.3 Periods of Carrier Wave to Measure

The carrier frequency is measured during the first pulse of the IR signal. The larger the number of periods used to measure, the more accurate the result is likely to be, however some signals have short initial pulses, so in some case it may be necessary for applications to reduce this value. Default value is 8 periods.

5.2.16.4 Length Measurement "Fuzz"

Due to the approximate nature of IR signal data, two supposedly identical pulse lengths will be slightly different. This attribute controls the size of the length "fuzz" or maximum difference that is accepted when evaluating whether two lengths are to be considered the same, i.e:

```
if ((length1 - length2 <= fuzz) {
                length2 = length1;
}
```

If the two values are not identical, then some primitive averaging takes place.

The units of this value are the same as the length array value units, which is counts of a 2MHz timer. To convert this to milliseconds, use the equation:

$$fuzz\_value\_in\_ms = \frac{counter\_value * 1000}{2MHz}$$

The default value is 112 (0.056mS).

5.2.16.5 Pause measurement timeout

The signal "pause" is the gap between the main signal and the repeat signal (or between repeat successive repeats). When sampling a signal, the *irNetBox* listens to this period of IR inactivity for a while, and then decides that if it goes on too long, it really must be the end of the signal. This parameter can be used to determine how long it waits before this timeout.

The units are a count at 2MHz, so to convert this value to ms, use the same equation as given above for the length "fuzz" value. The default value is 300,000, which corresponds to a time period of 150ms.

5.2.16.6 Minimum pause value

A very long OFF pulse within a signal could potentially be mistaken as a short pause. This parameter allows the application to set the minimum value the intra-signal pause can take (equivalent to the maximum value on OFF length can take).

The units are slightly indirect, being the value placed in the high byte of a 2MHz, incrementing timer. The equation below shows this a little more clearly:

$$\min\_pause\_in\_ms = \frac{65536 - (value << 8) * 1000}{2MHz}$$

The default value is 115, giving a millisecond result of 18mS.

5.2.17 [Message 33] Set the Signal Capture/Memory Parameters

This message sends a data block in the form given in the previous section to configure the memory in the irNetBox so that an IR signal can be downloaded.

5.2.18 [Message 34] Set All Outputs Using a Bit Mask (MK-II & III only)

If an IR signal is to be sent to multiple outputs, then it can be slow enabling all required ports. This command sets all outputs to the required state in a single operation.

Each port can be in one of 4 states:

Bit mask for a single port – 2 bits	
OFF	00
LOW power	01
MEDIUM power	10
HIGH power	11

The full data structure for all ports is 4 bytes, each byte containing the data for 4 output ports:

Full port bit mask structure															
byte 3				byte 2				byte 1				byte 0			
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

6 Asynchronous IR Output (MK III only)

The above method of IR signal output is synchronous, so however fast the hardware it will always be limited by the fact that an IR signal is sent for output and the irNetBox will only then respond once the output is complete. So to support concurrent output of different IR signals, asynchronous IR output was added as shown in the sequence diagram in Figure 3.

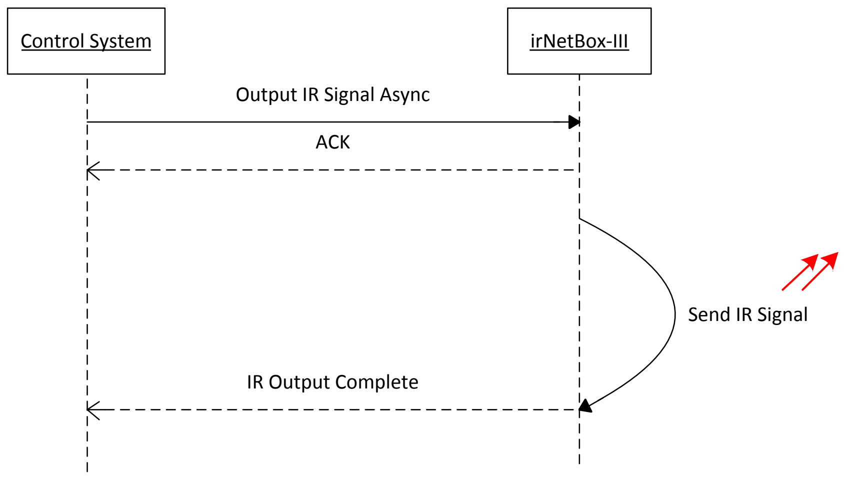


Figure 3. Basic async IR output operation

The control system sends the async IR output message and receives an ACK from the irNetBox-III. It can then continue with other tasks, and at some later point in time it will receive an IR output complete message.

To output two IR signals concurrently, the sequence shown in Figure 4 would typically take place.

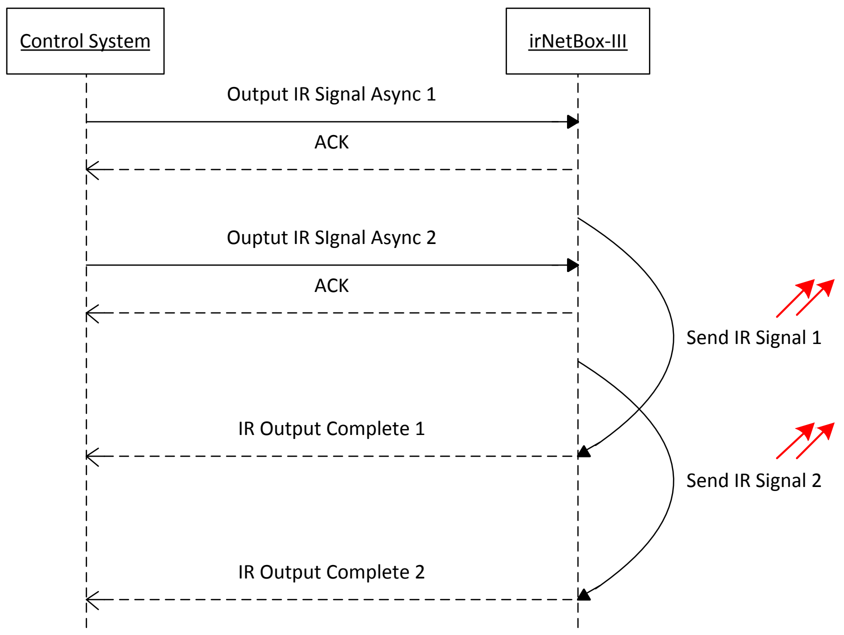


Figure 4. Overlapped output of two IR signals

Some points to note about this are:

- Each async output IR message contains all data necessary for output of that signal, including which ports are to be used at which power levels etc. This is described in detail in section 6.1.1.

- The time taken for communication is short compared with the usual time for IR output, so many async IR output commands can be sent before IR output completion messages start being sent back to the control computer.
- The IR Output Complete messages will not necessarily be returned in the same order as the async IR Output Signal message, the return order will also depend on the time taken to output each IR signal. As a result, async messages include a sequence number to correlate the returned completion message with the original async output message. See section 6.1.2 for more detail.
- An async IR Output message may result in a NACK rather than ACK if that output command can't be processed for any reason. An example of this is that one or more of the requested ports are busy, as shown in Figure 5.

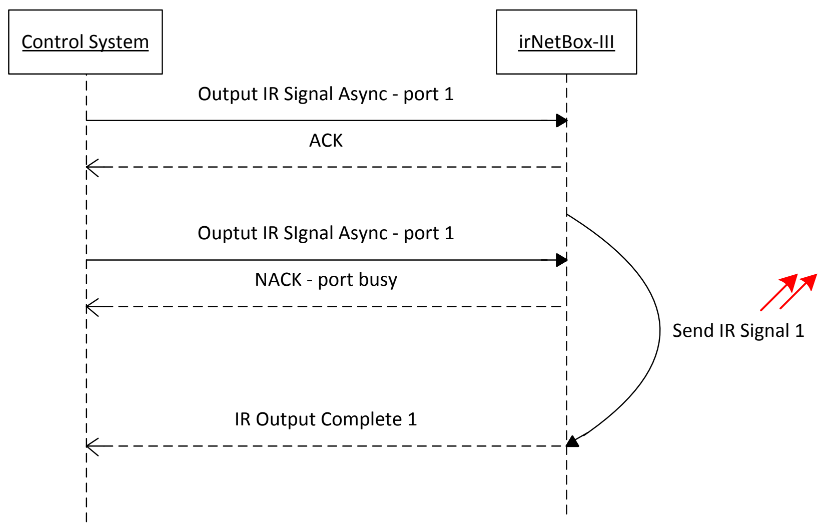


Figure 5. Attempt to send two async output commands to same port

6.1.1 [Message 48] Asynchronous IR Signal Output (MK-III only)

This command instructs the box to output the IR data on the given ports at the set power. The response of the irNetBox is to send back an immediate ACK or NACK, and if ACK'd, then send back an output complete message at a later point in time.

Structure of the asynchronous IR signal output data block (host → irNetBox-III)	
Message sequence number	ushort (little-endian)
Post signal delay (ms)	ushort (little-endian)
Port 1 output power level	byte
Port 2 output power level	byte
Port 3 output power level	byte
... for all 16 ports – a total of 16 bytes	
IR data to output	byte[]

6.1.1.1 Sequence Number

As the message response is asynchronous, the sequence number is used to pair returned messages (0x30 and 0x31) with the original outgoing message. Valid sequence number values are 0 to 65535.
NOTE: This is downloaded as a little-endian number (unlike values in the sync API), so on most x86/x64 based computers the bytes don't need to be swapped.

6.1.1.2 Post Signal Delay

The irNetBox-III is capable of sending IR signals asynchronously with almost no delay between them which can cause the receiving device difficulty in recognising the IR data. As a result a default delay of 100ms is set following the output of each IR signal, however the gap can be adjusted by setting this value, which is in ms. The maximum value is 10,000, i.e. 10s. If the value of 0 is set, then the default of 100ms is used.
NOTE: This is downloaded as a little-endian number (unlike values in the sync API).

6.1.1.3 Port Output Power Level

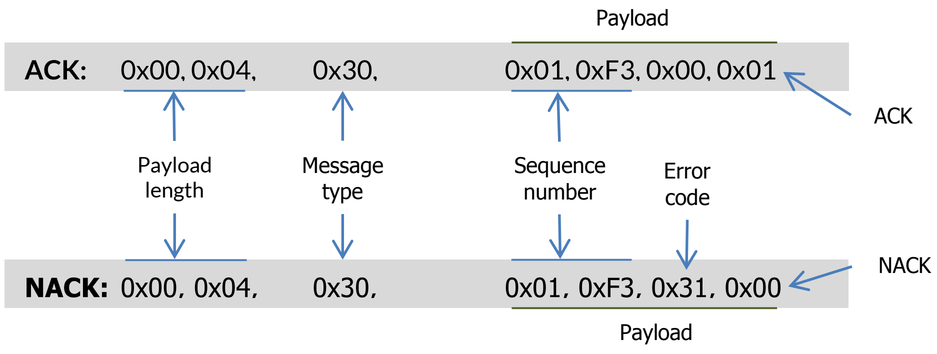
A value between 1 and 100 to indicate output power on that port. If the power level is 0 for a given port, then that port is not used in this command and is available for use by another output command.

6.1.2 [Message 48 contd] ACK/NACK Response to IR Signal Output (MK-III only)

This message is from the irNetBox-III to the host only and is used to indicate whether the async IR output command has been accepted or not. For example if one or more of the requested output ports is in use, then the command will be rejected. The response from the irNetBox-III is one ushort number containing the sequence number, a byte indicating whether the command is ACK'd or NACK'd and if NACK'd then the reason for this.

Asynchronous IR signal output ACK (irNetBox-III → host)	
Message sequence number	ushort (big-endian)
Error code	byte
ACK or NACK	byte

Where 1 is for ACK (command accepted) and 0 for NACK (command rejected). If a NACK is returned, the reason is given in the error code byte.


Payload
ACK: 0x00, 0x04, 0x30, 0x01, 0xF3, 0x00, 0x01
ACK
Payload length
Message type
Sequence number
Error code
NACK
NACK: 0x00, 0x04, 0x30, 0x01, 0xF3, 0x31, 0x00
Payload

The error codes that can be returned in a NACK are:

Error codes returned in message 48 (0x30) NACK		
Decimal	**Hex**	
49	0x31	The irNetBox-III is currently busy on one or more of the requested ports
50	0x32	The irNetBox-III processor message queue is full
51	0x33	IR signal modulation frequency is too low, i.e. less than 5KHz
52	0x34	IR signal modulation frequency is too high, i.e. greater than 490KHz
53	0x35	IR signal data section size is too large (max 2048 bytes)
54	0x36	Invalid signal data - too many EOS or EOR markers in the signal data
255	0x37	Too many length values in the IR signal data

6.1.3 [Message 49] Asynchronous IR Output Complete (MK-III only)

The message sent by the irNetBox-III when the asynchronous IR output command is complete. It will only be sent if the original IR Output message (48) was ACK'd, and contains four bytes of data, the first two being the message sequence number of the IR Output message as a ushort (little-endian).

Asynchronous IR output complete message (irNetBox-III → host)

Message sequence number	ushort (little-endian)
Reserved	byte[2]

As has been mentioned above, these messages will not be in the order of the original IR Output commands, but will instead be sent as soon as that particular IR signal output is complete, so are dependent upon the time taken by that IR signal.

7 Example Message Sequences Required to Perform Operations

To instruct the *irNetBox* to perform operations, sequences of operations are sometimes required.

7.1 Connecting to the irNetBox

When a TCP/IP connection is established to the irNetBox, there are a few operations that can be done at this stage to put it into its default operational state:

1) **Message 5: Power on the CPLD**
   In a standby state, to reduce power consumption the power to the CPLD device can be turned off. When the *irNetBox* is booted, the initialized state is that the CPLD has no power supplied, so before any IR input/output operations can be performed it must be powered-up.

2) **Message 7, Instruction 0: Reset the CPLD**
   Following power on, the CPLD should be in an initialized state where all IR outputs are off, but it is worth sending this instruction just in case.

3) **Message 7, Instruction 23: Set the red LEDs to reflect enabled IR outputs**
   This allows the user to observe the activity of the *irNetBox*. In this mode, a red LED on the front panel lights when the corresponding IR output has been enabled. When an IR signal is then transmitted, the red LED flashes off momentarily.

The above is just a recommendation, so does not have to be followed as given here.

7.2 Disconnecting from the irNetBox

Recommended operations are:

1) **Message 7, Instruction 0: Reset the CPLD**

2) **Message 6: Power off the CPLD**
   This ensures that while there is no active use of the *irNetBox*, it is in a slightly lower power state.

7.3 *Outputting* an IR Signal

Before starting this sequence of operations, ensure that the CPLD device is powered-on [Message 5]. The sequence of operations for IR signal output is:

1) **Message 16: Allocate memory for IR signal data. [First time only]**
   Instruct the *irNetBox* to allocate memory for receiving the signal data. The reason that this is a separate step is that the serial communication protocol between the XPort and the IR I/O processor has no flow control, and allocating RAM takes a relatively long operation so would cause data loss during the transfer if done during signal download.

2) **Message 7, Instruction 0: Reset the CPLD**
   This will disable all IR outputs, which is necessary before selecting the IR outputs required for this operation.

3) **Message 7, Instructions 3 to 17; Enable IR Outputs**

For each output to be enabled for this operation, this message should be sent with the appropriate instruction number.

4) **Message 33: Set the memory parameters to reflect that of the signal to be downloaded**
   This is only necessary if the default values are not being used. When the signal is downloaded as a block of data, it must fit exactly onto the footprint of the allocated memory on the *irNetBox*, which is determined by these parameters. If non-default values are being used, or have been previously set in the *irNetBox*, then it is necessary to use this message to sync the values on the *irNetBox* with those for the signal about to be downloaded. See section 5.2.8.

5) **Message 17: Download the IR Signal Data**

6) **Message 18: Output the IR Signal.**
   This causes the IR signal to be output via all enabled outputs. It can be called as many times as this signal is required to be output.

The sequence of instruction can of course be varied depending on the exact operation. For example, if signals are always being output to the same set of outputs, then there is no reason to reset the CPLD and re-enable the required outputs.

8 Resetting the XS1 Processor (MK-III only)

In the MK-III irNetBox, the main processor can be remotely reset by application software if needed. The Lantronix XPort network interface module has three general purpose IO (gpio) pins, and one of these is connected to XS1's reset circuitry.

<u>Note:</u> It is essential that all gpio pins are configured correctly for correct operation of the irNetBox, so care should be taken when writing code to change them. If they do become set incorrectly, they can be set back to the correct values for that particular irNetBox type by running any RedRat application (such as the IrNetBox Manager) and connecting to the box. In the background, the RedRat core library checks, and if necessary sets, the gpio pins on connection.

The port used for controlling the gpio pins is 0x77F0 (30704), and connections can be made to this port via TCP or UDP. For a detailed description of the command structure to read the state of the gpio pins and set to a new state, please see the section on the GPIO Interface in the ***XPort User Guide***, which can be downloaded from the Lantronix web site.

XPort GPIO Pin Settings				
	Direction	Active Level	MK-I and MK-II State	MK-III State
Pin 1	Output	High	Inactive	Inactive
Pin 2	Output	High	Active	Active
Pin 3	Output	High	Active	Inactive

The table above shows the gpio pin settings, and the last two columns the default state for irNetBox operation.

To reset the XS1 processor, pin 1 is set active for a short while. In detail:

1. Set pin 1 active
2. Wait 20ms
3. Set pin 1 inactive

9 An Introduction to Remote Control Signals

Although it is not necessary to understand a great deal about remote control signals to develop applications using RedRat products, some information on IR signals may make aspects of the API clearer, especially objects that represent the signals.

9.1 Modulated Remote Control Signals

The vast majority of remote control signals are in this family, having a carrier wave usually (though not always) in the frequency range 36kHz to 40kHz. Figure 6 shows the layers in such a signal. The RedRat does not attempt to interpret or discover the coding scheme used in the signal (e.g. shift/biphase coded, space coded, RC5, RCMM, REC-80 etc.) as knowledge of this is not needed for recognition or reproduction.

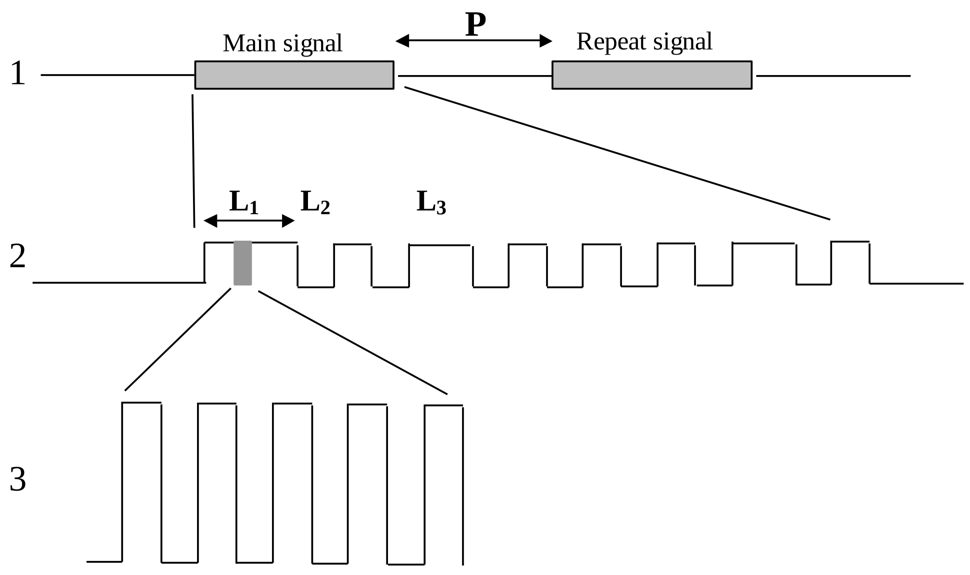


Figure 6. Layers in a Modulated Remote Control Signal.

9.1.1 Layer 1 – Main and Repeat Signals

These two sections to a signal are most commonly used to indicate the duration of a button press on a remote; the main signal being sent once and the repeat signal sent repeatedly until the button is released. On some remotes the repeat signal is identical to the main signal, but it can often be a different. As a result, the RedRat stores them as separate signal sections regardless of whether they are identical or not.

There are of course variations, such as a 3 part signal (button down, hold and release) or repeating the whole main/repeat section. The value **P** is the inter-packet pause length.

9.1.2 Layer 2 – Signal Envelope

This is the basically the sequences of pulses and gaps that carry the signal information. When the RedRat samples the signal, it builds up an alphabet of the *lengths* of the pulses and gaps, L_1, L2, L_3 etc. The actual signal data is then a series of numbers which are a lookup into the length array, i.e. 123333 is a sequence of length 1, length 2, length 3, length 3 etc.

9.1.3 Layer 3 – Carrier/Modulation Frequency

Each IR pulse is actually a rapidly switching signal, usually around the 36kHz to 40kHz allowing detectors to filter out background IR from this signal, so giving good transmission range and reliability. Standard remote control detectors strip out the carrier frequency and in doing so also alter the actual length values. This does not impact signal recognition, but in many cases are not sufficiently accurate for reliable reproduction. The RedRat samples the raw signal, giving accurate L values and a good measurement of the carrier frequency.

9.2 IrDA-Like Signals

Some set-top boxes use an IrDA-like remote control transmission protocol which offers some advantages over the modulated signal type described above, such as a higher data rate, supporting multiple handsets and time-stamping signals. Figure 7 shows part of such a signal, comprising a series of sub-packets (15 at least) separated by quite large gaps.

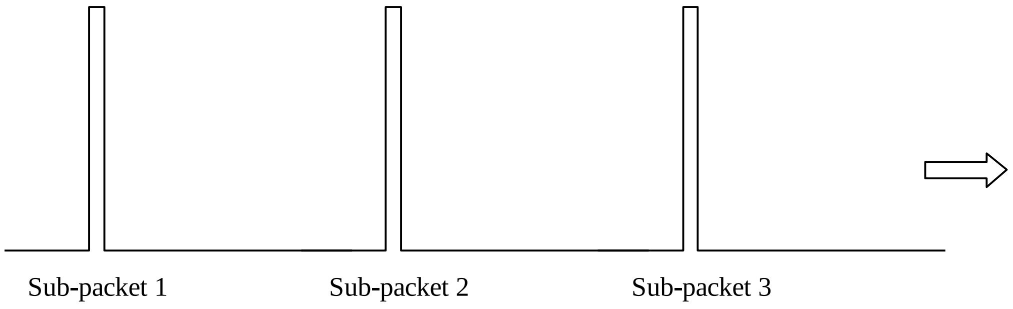


Figure 7. Part of an IrDA-like Signal

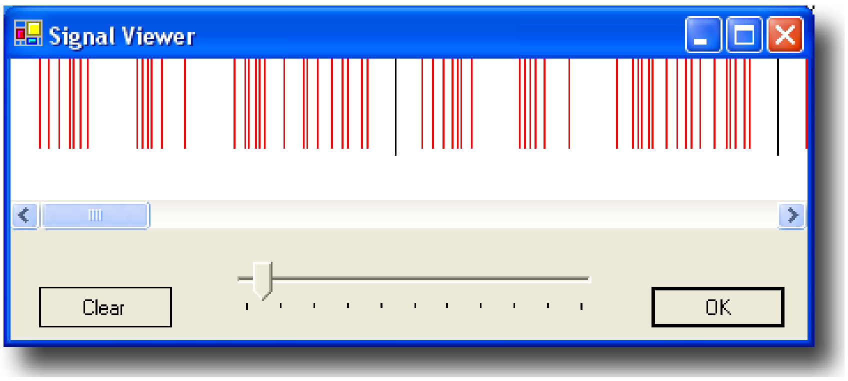


Figure 8. Sub-packet structure of an IrDA-like Signal.

Each sub-packet is a series of very short pulses (a couple of uS) with varying separation. Figure 8 shows the structure of the first two sub-packets of an signal, the large inter-sub-packet pause having been removed and replaced by the black vertical lines.

9.3 Flash-Code Signals

These signals are not common, and generally found in older equipment. They can be considered as very similar to modulated signals, except that the IR output when on is not modulated, rather constant.

RedRat products do not currently contain the firmware code to output these kind of signals, however this will follow soon.

10 Understanding the RedRat XML Signal Data Format

The Signal Database utility manages sets of IR signals and stores them in an XML file. If application development using the *irNetBox* does not use the RedRat SDK, then it may be necessary to convert from data from the XML files to binary signal data blocks for downloading and output. The section describes the XML format for a *ModulatedSignal* object.

```
<IRPacket xsi:type="RedRat3ModulatedSignal">
    <Name>Up</Name>
    <UID>pWurW50QdU+T5yX9TK0c3A==</UID>
    <ModulationFreq>37383</ModulationFreq>
    <Lengths>
       <double>9.0305</double>
       <double>4.7725</double>
       <double>0.494</double>
       <double>1.758</double>
       <double>0.622</double>
       <double>1.6644999999999999</double>
```

```
      </Lengths>
      <SigData>AAECAwIEAgQCBAIEAgQCAwIDAgQCBAIDAgQC==</SigData>
      <NoRepeats>1</NoRepeats>
      <IntraSigPause>41.573</IntraSigPause>
      <ToggleData>
         <ToggleBit>
            <bitNo>2</bitNo>
            <len1>0</len1>
            <len2>1</len2>
         </ToggleBit>
         <ToggleBit>
            <bitNo>4</bitNo>
            <len1>1</len1>
            <len2>0</len2>
         </ToggleBit>
      </ToggleData>
   </IRPacket>
```

The RedRat signal database utility can be used to generate signal data blocks from XML signal databases for reading into an application to download – see section 11.

10.1 IRPacket

All IR signal data is represented in the RedRat SDK by *IRPacket* objects which have a certain subclasses, depending on signal and hardware type. The example shown is modulated signal data that has been input by the RedRat3. The *irNetBox* signal input and output uses the same signal representation as the RedRat3.

10.2 Name

The signal name as displayed in the Signal DB utility. This is metadata only, i.e. it is not needed for signal data transfer to/from the device.

10.3 UID

Unique identifier for the *IRPacket* object, given when the signal is first input. The is metadata so is not required for data transfer to/from the device.

10.4 ModulationFreq

The modulation/carrier frequency in Hz.

10.5 Lengths

The length array data in ms.

10.6 SigData

This is a byte array converted for storage in a text file. The conversion/storage format is base64 in the XML file. Once converted back from base64, the byte array should contain a set of values such as 0, 1, 2, 2, 3, 4, 3, 4, 5. The binary value 127 is used as the end of signal marker, used at the end of both the main and repeat signal sections.

Note: The SigData code snippet in the above XML sample has been truncated so is not a valid set of data.

10.7 NoRepeats

The number of times the repeat signal is output.

10.8 IntraSigPause

When a signal is output many times, the pause time between the signal output is given in ms by this value.

10.9 ToggleData

Some signals have a few bits that alternate each time the signal is output, and this information is held in this section, comprising one or more *ToggleBit* blocks.

10.9.1 ToggleBit

Represents one bit that varies in a signal, the number of that "bit" in the *bitNo* field. In effect, this is the offset in the signal data array. The two values given in the *len1* and *len2* fields are the values to alternately place into the signal data array each time the signal is output.

Note: Often it is not necessary to use the toggle bits on signal output as the A/V device being controlled will respond regardless.

11 Using the Signal Database Utility for Creation of Signal Data Memory Blocks

The RedRat Signal Database Utility is the main tool for IR signal management. From version 1.15 onwards, it can be used to convert XML signal data for a device/remote to the signal data blocks that can be directly downloaded. For ease of storage and manipulation, the binary data is written to a file in ASCII hex representation, i.e. the byte value 255 is represented as "FF".

11.1 Exporting a Device in IRNetBox Format

A "device" is a single piece of audio visual equipment in utility terminology, so there is usually one remote for each device. To export:

- Select a device/remote in the main window
- From the *Edit* menu, go to *Export Device/Remote* and select *IrNetBox Signal Data*.

11.2 The Exported Data File Format

Each IR signal in the dataset is placed on a single line, with the following fields:

<signal name> <max_num_lengths> <byte_array_in_ascii_hex>

The actual data string is a byte array, each byte having been converted to ASCII Hex, so FF is a byte with value 255 etc. This data is to be read by an application program and converted back to a byte array, it should simply be used as the block of data in the message 17 to the *irNetBox*.

The byte array is created to be placed into the RAM signal footprint allocated in the *irNetBox*. The exact size of this footprint depends on the parameter *Maximum Number of Lengths* (see section 5.2.16), i.e. the maximum number of length values a signal can have. The value of this parameter used by the Signal DB utility is placed before the data string, so the application program should also read this value and ensure that the *irNetBox* has been set with this value before the signal is downloaded – see section 5.2.8.

By default, this value is 16 which should be large enough for most signals, so currently the Signal DB utility creates the signal binary data assuming this value.