



**Connect Tech Inc.**  
*Industrial Strength Communications*

---

# Blue Heat/Net

## Protocol Manual



**Connect Tech, Inc.**  
**42 Arrow Road**  
**Guelph, Ontario**  
**Canada, N1K 1S6**  
**Tel: 519-836-1291**  
**800-426-8979 (North America only)**  
**Fax: 519-836-4878**  
**Email: [sales@connecttech.com](mailto:sales@connecttech.com)**  
**support@connecttech.com**  
**URL: <http://www.connecttech.com>**

**CTIM-00049 Revision 1.04 - January 5, 2009**

## Copyright Notice

The information contained in this document is subject to change without notice. Connect Tech, Inc. shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material. This document contains proprietary information that is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced, or translated to another language without prior written consent of Connect Tech, Inc.

Copyright 2008 by Connect Tech, Inc.

Revision 1.04 - January 5, 2009

Date Published: January 5, 2009

## Trademark Acknowledgement

Connect Tech, Inc. acknowledges all trademarks, registered trademarks and/or copyrights referred to in this document as the property of their respective owners.

Not listing all possible trademarks or copyright acknowledgements does not constitute a lack of acknowledgement to the rightful owners of the trademarks and copyrights mentioned in this document.

---

## Revision History

### **Revision 1.04    January 5, 2009**

- > Added Conventions section.

### **Revision 1.03    October 28, 2008**

- > Added Network Byte Ordering section.
- > Updated protocol status value tables.

### **Revision 1.02    October 6, 2008**

- > Removed Protocol Versioning information and added new supporting messages.
- > Added Real Time Clock tutorial.

### **Revision 1.01    September 24, 2008**

- > Release

### **Revision 1.00    April 23, 2008**

- > Preliminary Release



# Contents

- 1 Introduction ..... 2
  - 1.1 Supported Products and Platforms .....2
  - 1.2 Contact Information .....2
  - 1.3 Conventions .....3
    - Notation .....3
    - Data Types .....3
  - 1.4 Obtaining the Protocol .....3
- 2 The Protocol ..... 4
  - 2.1 Overview .....4
  - 2.2 Message Format .....5
  - 2.3 Response Format .....6
  - 2.4 Data Format.....6
    - Data Modes.....6
    - Directors.....7
  - 2.5 Protocol Usage .....7
    - General Messages .....7
    - Port Messages .....8
    - Data.....8
    - Protocol Versions .....9
  - 2.6 TCP Port Layout.....9
  - 2.7 Network Byte Ordering .....9
- 3 Protocol Reference ..... 11
  - 3.1 Message Summary..... 12
  - 3.2 Message Detail ..... 13
    - CANCEL\_RTC\_MPO..... 13
    - CLOSE\_PORT..... 14
    - EVENT\_OCCURRED ..... 14
    - EVENT2\_OCCURRED..... 16
    - GET\_BHN\_INFO ..... 17
    - GET\_DATA\_MODE..... 18
    - GET\_EVENT..... 19



GET_EVENT2.....	20
GET_FIFO_CONTROL.....	21
GET_LINE.....	22
GET_LINE2.....	23
GET_LINE_STATE.....	24
GET_MSR.....	25
GET_PORTCAP.....	26
GET_PROTO_SUPPORT.....	27
GET_PROTO_VER.....	28
GET_PSON.....	29
GET_RTC_ALARM.....	30
GET_RTC_CLOCK.....	31
GET_RTC_GATE.....	32
GET_RTC_MODE.....	32
OPEN_PORT.....	33
PING.....	34
PURGE.....	35
RESET_BHN.....	36
RESUME_TX.....	36
SEND_BREAK.....	37
SEND_IMMEDIATE.....	38
SEND_XCHAR.....	38
SET_DATA_MODE.....	39
SET_DTR.....	40
SET_EVENT.....	41
SET_EVENT2.....	42
SET_FIFO_CONTROL.....	45
SET_LINE.....	47
SET_LINE2.....	49
SET_RTC_ALARM.....	61
SET_RTC_CLOCK.....	63
SET_RTC_GATE.....	64
SET_RTC_MODE.....	65
SET_RTS.....	66
SET_RTS_DTR.....	67
START_STOP_BREAK.....	67
SYNC_HUNT.....	68
3.3 Director Summary.....	69
3.4 Director Detail.....	70



---

DATA .....	70
EVENT2 .....	70
END_OF_FRAME .....	71
START_OF_FRAME .....	71
4 Index .....	73



# 1 Introduction

1.1 Supported Products and Platforms .....	2
1.2 Contact Information .....	2
1.3 Conventions .....	3
Notation .....	3
Data Types .....	3
1.4 Obtaining the Protocol .....	3

## 1.1 Supported Products and Platforms

This documentation applies to the following Connect Tech products:

### **Connect Tech Products**

- The Blue Heat/Net family of products
  - Blue Heat/Net 2
  - Blue Heat/Net 4 or 8
  - Blue Heat/Net 16
  - Blue Heat/Net Sync

## 1.2 Contact Information

We offer three ways for you to contact us, by telephone or facsimile (FAX), by email or internet, or by mail or courier.

### **Telephone/Facsimile**

Technical Support representatives are ready to answer your call Monday through Friday, from 8:30 a.m. to 5:00 p.m. Eastern Standard Time. Our numbers for calls are:

<b>Telephone</b>	800-426-8979 (North America only)
	519-836-1291 (Live assistance available 8:30 a.m. to 5:00 p.m. EST, Monday to Friday)
<b>Facsimile</b>	519-836-4878 (on-line 24 hours)

### **Email/Internet**

You may contact us through the Internet. Our email and URL addresses are:

<b>Email</b>	<a href="mailto:sales@connecttech.com">sales@connecttech.com</a>
	<a href="mailto:support@connecttech.com">support@connecttech.com</a>
<b>Web</b>	<a href="http://www.connecttech.com">http://www.connecttech.com</a>



### Mail/Courier

You may contact us by letter. Our mailing address for correspondence is:

<b>Mail</b>	Connect Tech Inc. Technical Support 42 Arrow Road Guelph, Ontario Canada N1K 1S6
-------------	--

## 1.3 Conventions

### Notation

This document uses the following notational conventions:

**file names**

filesystem paths

*special names*

**CONSTANT NAME** references such as messages, directors, or constants

code blocks

**structure name** references

**structure member** references

hyperlinks

### Data Types

The data types used in this document are generic operating system independent names.

Table 1-1 lists the generic data types and their storage characteristics.

Type	Description
int8	Signed 8-bit integer.
int16	Signed 16-bit integer.
int32	Signed 32-bit integer.
uint8	Unsigned 8-bit integer.
uint16	Unsigned 16-bit integer.
uint32	Unsigned 32-bit integer.

Table 1-1

## 1.4 Obtaining the Protocol

The protocol definition is available as a C header file and is part of the Blue Heat/Net SDK.

This header file, **BHN\_protocol.h**, can be found in the directory

<sdk\_root>/vendors/ConnectTech/include, where <sdk\_root> is an extracted Blue Heat/Net SDK.



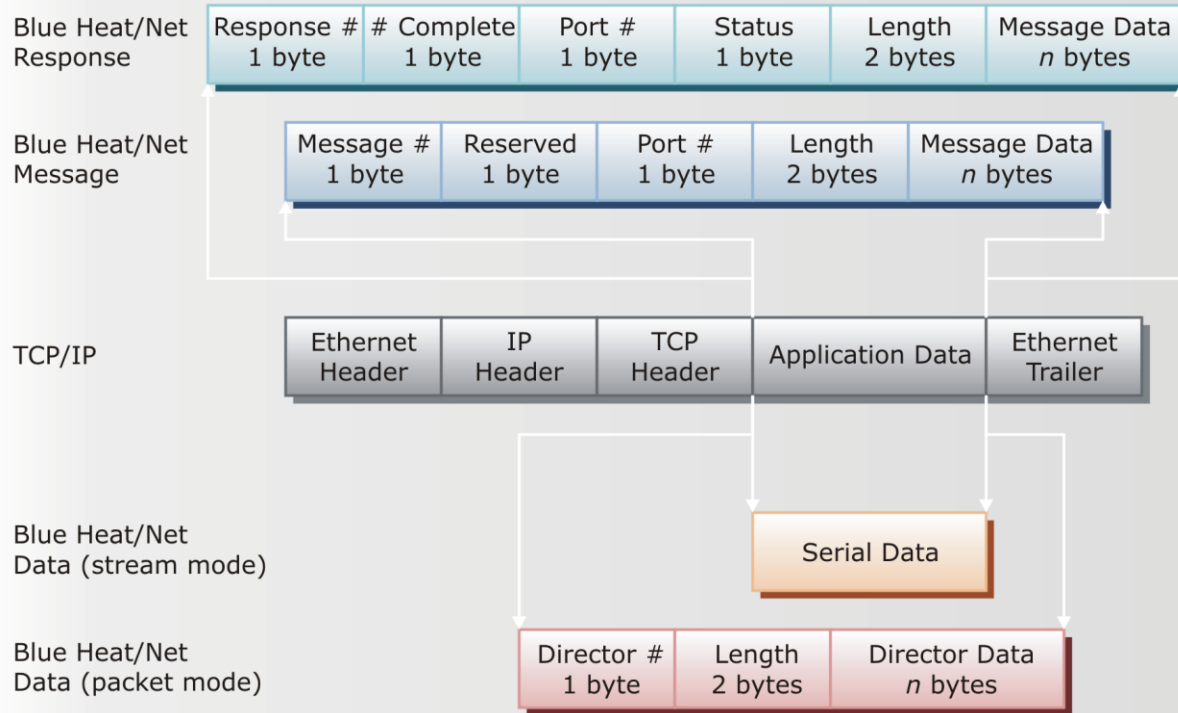
## 2 The Protocol

2.1 Overview .....	4
2.2 Message Format .....	5
2.3 Response Format .....	6
2.4 Data Format.....	6
Data Modes.....	6
Directors .....	7
2.5 Protocol Usage .....	7
General Messages .....	7
Port Messages .....	8
Data .....	8
Protocol Versions .....	9
2.6 TCP Port Layout .....	9
2.7 Network Byte Ordering .....	9

### 2.1 Overview

The Blue Heat/Net protocol uses a series of TCP connections (channels) to communicate between the host and Blue Heat/Net. A Blue Heat/Net is controlled by sending packets of data called *messages* from the host to the Blue Heat/Net via TCP. The Blue Heat/Net can also send notifications back to the host using *messages*. Most *messages* result in feedback being sent in the form of *responses*, and are really just a specific type of *message*. While *messages* and *responses* are used for controlling a Blue Heat/Net, serial data is communicated in one of two different modes. The first mode is *stream mode*, where the serial data is not encapsulated in any type of protocol and is just carried by TCP directly. The second mode is *packet mode*, where, unlike *stream mode*, the data is encapsulated in a protocol. Serial data transferred in *packet mode* is split into frames of data and each frame is split into 1 or more *directors*.

The remainder of this chapter details the *messages*, *responses*, and *directors* used to communicate with a Blue Heat/Net. [Figure 2-1](#) gives a visual representation of these concepts.



**Note:** more than one *message*, *response*, or *director* could be in a given TCP packet. Also, a single *message*, *response*, or *director* could be split among multiple TCP packets.

Figure 2-1

## 2.2 Message Format

Each protocol message, contains a header and optional data. A message header consists of a 1 byte *message number*, a 1 byte reserved area, a 1 byte *port number*, and a 2 byte *length*. Following the message header is optional *message data* whose size depends on the *length* specified in the message header.

### Message Number

The first byte of a message header is the *message number*. The *message number* indicates the message’s intended operation and determines the format of the *message data*. See section 3, [Protocol Reference](#), for a list and description of the available *message numbers*.

### Port Number

The third byte of a message header is the zero-based *port number*. The *port number* indicates which port on the Blue Heat/Net device that the message applies to. In section 3, [Protocol Reference](#), the usage requirements for the *port number* is specified for each individual message.

### Length

The fourth and fifth bytes of a message header are the size of the optional *message data*.

### Message Data

Following a message header is the message’s optional data. The size and format of this data is dependent on the *message number* contained in the message header. See section 3, [Protocol Reference](#), for a list and description of the available messages and their corresponding data.



## 2.3 Response Format

Each protocol response contains a header and optional data. The response header is 6 bytes long and consists of a 1 byte *response number*, a 1 byte *completed message number*, a 1 byte *port number*, a 1 byte *status*, and a 2 byte *length*. Following the response header is optional *response data* whose size depends on the *completed message number*.

### Response Number

The first byte of a response header is the *response number*. The *response number* indicates the response's type. There is currently only one valid *response number*, **CMD\_DONE**, which has a value of 128 (0x80).

### Completed Message Number

The second byte of a response header is the *completed message number*. A response is the result of a message and the *completed message number* specifies which message this response is related to.

### Port Number

The third byte of a response header is the zero-based *port number*. The *port number* indicates which port on the Blue Heat/Net device that the response applies to. Not all responses actually need a correct value for the *port number*; this is because the TCP port implies the *port number*.

### Status

The fourth byte of a response header is the *status*. The response *status* indicates the results of the corresponding message. Refer to the specific message in section 3, [Protocol Reference](#) for details about the status values and meanings.

### Length

The fifth and sixth bytes of a response header are the size of the optional *response data*.

### Response Data

If the *length* specified in the response header is greater than zero, the response header is followed by the *response data* area of that size. The format of the *response data* is dependent on the *completed message number* specified in the response header. See section 3, [Protocol Reference](#), for a list and description of the available messages and their corresponding *response data*.

## 2.4 Data Format

While message and response channels are used to configure and control the Blue Heat/Net, data channels are used to transmit and receive the actual serial data.

### Data Modes

A data channel can operate in one of two data modes. The first data mode, *stream mode*, is a simple, continuous stream of serial data only. The second data mode, *packet mode*, encapsulates the serial data in a protocol similar to that of the message and response channels.

#### Stream Mode

*Stream mode* is the default data mode for a data channel. In *stream mode* the serial data bits are transmitted and received on a data channel in the form of a continuous stream of data. This stream of data does not include control bits such as the start, parity, and stop bits. Host software can transmit data by just writing to the data channel and similarly data received by the Blue Heat/Net can be retrieved by reading from the data channel.



### **Packet Mode**

In *packet mode* the serial data bits are encapsulated in a protocol consisting of directors. Similar to *stream mode*, the data contained in each director does not include control bits such as the start, parity, and stop bits. However, using *packet mode* allows other types of information besides data to be transported in the data channel while remaining synchronized with the data.

### **Directors**

A *director* is a block of data used in the *packet mode* protocol. The term *director* is used to help differentiate from other terms such as packets (TCP), messages, and frames. Not only does a *director* contain serial data, it can also contain notification of events and data status information. Each *director* consists of a 1 byte *director number*, 2 byte *length*, and an optional *data area*.

#### **Director Number**

The first byte of a *director* is the *director number*. The *director number* indicates the director's type and determines the format of the *director data*. See section 3, [Protocol Reference](#), for a list and description of the available *director numbers*.

#### **Length**

The second 2 bytes of a *director* is the size of the *director data*.

#### **Director Data**

If the *length* specified in the *director* is greater than zero, the director header is followed by the *director data* of that size. The format of the *director data* is dependent on the *director number*. See section 3, [Protocol Reference](#), for a list and description of the available directors and their corresponding *director data*.

## **2.5 Protocol Usage**

The Blue Heat/Net protocol is used by establishing one or more TCP connections (channels) depending on the desired task, while certain channels may only be established after other requirements have been met. This section deals with the details of setting up and tearing down channels, using the protocol, and protocol versioning. Refer to the host application's operating system documentation for information on how to create, work with, and destroy TCP connections.

The Blue Heat/Net waits for incoming TCP connections on at least two TCP ports; the general message channel and one message channel for each serial port. The Blue Heat/Net does not wait for incoming TCP connections for a data channel until after the corresponding serial port has been opened using the **OPEN\_PORT** message. Refer to [Table 2-2](#) for the default TCP ports used for each channel.

### **General Messages**

The general message channel can be used to access device-wide settings and information. A Blue Heat/Net will accept more than one connection to the general message channel, allowing more than one host application to access the device at the same time. A host application first creates a TCP connection to the general message channel's TCP port and sends messages by writing the appropriate message headers and data. Responses will be sent back through the TCP connection and are read by the host application. A connection to the general message channel can be torn down at any point by destroying the TCP connection from the host application.



## Port Messages

The port message channel can be used to access port-specific settings and information. Unlike the general message channel, a port message channel will only accept one connection at a time. However, like the general message channel, a host application first creates a TCP connection to the general message channel's TCP port and sends messages by writing the appropriate message headers and data. Responses will be sent back through the TCP connection and are read by the host application. If the data channel corresponding to a port message channel has not yet been opened then the host application can tear down the port message channel at any point by destroying the TCP connection. If the data channel has been opened, it is recommended that the data channel be torn down before the port message channel.

## Data

The port *message* **OPEN\_PORT** causes the Blue Heat/Net to listen for incoming TCP connections on the port's corresponding data channel. After sending the **OPEN\_PORT message**, the host application should wait to receive the **OPEN\_PORT response**, checking the status for success or failure. Upon receiving a successful **OPEN\_PORT response**, the host application may then proceed to make a new TCP connection to the data channel's TCP port. Once the host application has finished opening the port, any *messages* that require the port to be open will then function. When the host application is done using the serial port it should tear down the data channel by destroying the TCP connection and then sending a **CLOSE\_PORT message** to the port message channel.

### Stream Mode

To be completed.

### Packet Mode

Once a TCP connection has been made to the data channel that is operating in *packet mode*, the Blue Heat/Net is ready to send and receive *directors*. For the purposes of transferring serial data 4 *directors* are used; **START\_OF\_FRAME**, **DATA**, **END\_OF\_FRAME**, and **EVENT2**.

All transferred serial data is encapsulated in a frame, which is delimited by **START\_OF\_FRAME** and **END\_OF\_FRAME directors**. One or more **DATA directors** can come after a **START\_OF\_FRAME director** but before the **END\_OF\_FRAME director**. The **EVENT2 directors** may be sent from the Blue Heat/Net at any time, however if the **EVENT2 director** contains any data it will come after a **START\_OF\_FRAME director** and before an **END\_OF\_FRAME director**.

The **START\_OF\_FRAME**, **END\_OF\_FRAME**, and **EVENT2 directors** have optional serial data. This allows the serial data to be transferred in many different arrangements. For example, **Error! Reference source not found.** shows some different ways that 75 bytes of serial data could be transferred using *directors*.

The Blue Heat/Net will attempt to buffer a transmit frame as long as possible, until receiving an **END\_OF\_FRAME director**. If the Blue Heat/Net runs out of buffer space before receiving an **END\_OF\_FRAME director**, the current frame will begin transmitting in order to make room for the remaining *directors*.



START_OF_FRAME	DATA <sup>1,2</sup>	END_OF_FRAME
75 bytes		0 bytes
25 bytes		50 bytes
0 bytes		75 bytes
25 bytes	25 bytes	25 bytes
0 bytes	75 bytes	0 bytes

Table 2-1

## Protocol Versions

The Blue Heat/Net protocol has multiple versions, identified by a version number. Each version of the protocol contains a set of supported *messages*, *responses*, and *directors*. In order to determine the protocol version support of a Blue Heat/Net, use the **GET\_PROTO\_SUPPORT** message. However, the **GET\_PROTO\_SUPPORT** message is only supported on Blue Heat/Net devices that are running firmware version 1.33 or greater. For devices running firmware older than version 1.33, use the **GET\_PROTO\_VER** message.

If you are having trouble with the protocol version *messages* or have any questions please contact [Connect Tech support](#).

## 2.6 TCP Port Layout

The Blue Heat/Net protocol uses a number of different TCP ports for communications. The port numbers used are based on the Blue Heat/Net device's configurable base TCP port. The default base TCP port is 49152 (0xC000). Table 2-2 shows the layout of the TCP ports.

Base TCP Port Offset	Default TCP Port	Use
0	49152 (0xC000)	General message/response channel
2	49154 (0xC002)	Port 1 message/response channel
3	49155 (0xC003)	Port 1 data channel
4	49156 (0xC004)	Port 2 message/response channel
5	49157 (0xC005)	Port 2 data channel
6	49158 (0xC006)	Port 3 message/response channel
7	49159 (0xC007)	Port 3 data channel
...	...	...

Table 2-2

## 2.7 Network Byte Ordering

When working with network applications it is important that byte ordering be taken into account. The standard byte ordering in network applications is big-endian and host systems must ensure that they perform the appropriate byte-swapping on any message, response, or director structure member that is more than one byte. For example, on a little-endian host system (i.e. Intel x86) a 32-bit integer's byte ordering must be reversed before sending it to a Blue Heat/Net or after receiving it from a Blue Heat/Net.

Most operating systems provide functions to transform multi-byte values from host byte order to network byte order. The common names of these functions are listed in Table 2-3. An example of preparing and sending a **SET\_EVENT2** message is shown in Listing 2-1.

<sup>1</sup> Sending a DATA director of zero length is not legal.

<sup>2</sup> More than one DATA director could be used to break up the serial data further.



Function	Description
htons	Convert a 16-bit number from host byte order to network byte order.
htonl	Convert a 32-bit number from host byte order to network byte order.
ntohs	Convert a 16-bit number from network byte order to host byte order.
ntohl	Convert a 32-bit number from network byte order to host byte order.

**Table 2-3**

```
struct bhn_prot_cmd  msg;

msg.pc_cmd.cmd = SET_EVENT2;
msg.pc_cmd.sequence = 0;
msg.pc_cmd.port = 1;
msg.pc_cmd.length = htons(BHN_EVENT2_SZ);
msg.pc_event2.events = htons(EVENT_OE | EVENT_PE | EVENT_FE);
msg.pc_event2.events2 = htonl(EVENT2_ALL);
send(msg_socket, (char*)&msg, SET_EVENT2_CMD_SZ, 0);
```

**Listing 2-1**



### 3 Protocol Reference

3.1 Message Summary .....	12
3.2 Message Detail .....	13
3.3 Director Summary .....	69
3.4 Director Detail .....	70



### 3.1 Message Summary

Message Number	Message Name	Message Data Size <sup>1</sup>	Response Data Size <sup>1</sup>	Protocol Version <sup>2</sup>	Usage <sup>3</sup>		
					G	P	A S
1 (0x01)	<b>PING</b>	0	0	9+	G		
2 (0x02)	<b>OPEN_PORT</b>	1	0	9+		P	A S
3 (0x03)	<b>CLOSE_PORT</b>	0	0	9+		P	A S
7 (0x07)	<b>PURGE</b>	1	0	9+		P	A S
8 (0x08)	<b>SET_LINE</b>	26	0	9+		P	A S
9 (0x09)	<b>GET_LINE</b>	0	26	9+		P	A S
12 (0x0C)	<b>GET_LINE_STATE</b>	0	1	9+		P	A S
13 (0x0D)	<b>SEND_BREAK</b>	1	N/A	9+		P	A
14 (0x0E)	<b>START_STOP_BREAK</b>	1	0	9+		P	A
15 (0x0F)	<b>SET_RTS</b>	1	0	9+		P	A S
16 (0x10)	<b>SET_DTR</b>	1	0	9+		P	A S
19 (0x13)	<b>RESUME_TX</b>	0	N/A	9+		P	A
22 (0x16)	<b>GET_BHN_INFO</b>	134	319	9+	G	P	
28 (0x1C)	<b>RESET_BHN</b>	0	N/A	9+	G		
30 (0x1E)	<b>SEND_IMMEDIATE</b>	1	N/A	9+		P	A
31 (0x1F)	<b>SET_FIFO_CONTROL</b>	8	8	9+		P	A
32 (0x20)	<b>SET_EVENT</b>	2	0	9+		P	A S
33 (0x21)	<b>GET_EVENT</b>	0	2	9+		P	A A
34 (0x22)	<b>EVENT_OCCURRED</b>	19	N/A	9+		P	A S
36 (0x24)	<b>GET_MSR</b>	0	1	9+		P	A
37 (0x25)	<b>SEND_XCHAR</b>	1	0	9+		P	A
38 (0x26)	<b>SET_RTS_DTR</b>	2	0	9+		P	A S
39 (0x27)	<b>GET_PORTCAP</b>	0	16	9+	G	P	A S
41 (0x29)	<b>GET_FIFO_CONTROL</b>	8	8	9+		P	A
42 (0x2A)	<b>GET_PSON</b>	0	32	9+	G		
46 (0x2E)	<b>GET_PROTO_VER</b>	0	16	9+	G	P	
47 (0x2F)	<b>GET_PROTO_SUPPORT</b>	0	2	10+		P	
50 (0x32)	<b>SET_DATA_MODE</b>	1	0	10+		P	A S
51 (0x33)	<b>GET_DATA_MODE</b>	0	1	10+		P	A S
52 (0x34)	<b>SYNC_HUNT</b>	0	0	10+		P	A S
53 (0x35)	<b>SET_RTC_CLOCK</b>	3	0	10+		P	A S
54 (0x36)	<b>GET_RTC_CLOCK</b>	0	3	10+		P	A S
55 (0x37)	<b>SET_RTC_ALARM</b>	6	0	10+		P	A S
56 (0x38)	<b>GET_RTC_ALARM</b>	0	6	10+		P	A S
57 (0x39)	<b>SET_RTC_GATE</b>	1	0	10+		P	A S
58 (0x3A)	<b>GET_RTC_GATE</b>	0	1	10+		P	A S
59 (0x3B)	<b>GET_RTC_MODE</b>	0	1	10+		P	A S

<sup>1</sup> Message and response data sizes are in bytes.

<sup>2</sup> Protocol version refers to the earliest version that the message is supported. Versions earlier than 9 are no longer used and this document does not cover them.

<sup>3</sup> Usage has the following coding; G=General message, P=Port message, A=Async, S=Sync



Message Number	Message Name	Message Data Size <sup>1</sup>	Response Data Size <sup>1</sup>	Protocol Version <sup>2</sup>	Usage <sup>3</sup>
60 (0x3C)	<b>SET_RTC_MODE</b>	1	0	10+	P A S
61 (0x3D)	<b>CANCEL_RTC_MPO</b>	0	0	10+	P A S
62 (0x3E)	<b>SET_LINE2</b>	256	4	10+	P A S
63 (0x3F)	<b>GET_LINE2</b>	0	256	10+	P A S
65 (0x41)	<b>SET_EVENT2</b>	6	0	10+	P A S
66 (0x42)	<b>GET_EVENT2</b>	0	6	10+	P A S
67 (0x43)	<b>EVENT2_OCCURRED</b>	23	N/A	10+	P A S

Table 3-1

## 3.2 Message Detail

### CANCEL\_RTC\_MPO

The **CANCEL\_RTC\_MPO** message is used to cancel the minute pulse once mode.

#### Message

*Message Number*

61 (0x3D)

*Port Number*

Number of port in which to cancel the minute pulse once mode.

*Data Length*

0 bytes

#### Response

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>CANCEL_RTC_MPO</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>CANCEL_RTC_MPO</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>CANCEL_RTC_MPO</b> message.

*Data Length*

0 bytes

#### Remarks

The minute pulse once mode is enabled by sending a **SET\_RTC\_MODE** message with **RTC\_MINUTE\_ONCE** set. **CANCEL\_RTC\_MPO** can be used to disabled **RTC\_MINUTE\_ONCE**.



## Requirements

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

## See Also

[SET\\_RTC\\_MODE\\_OPEN\\_PORT](#)

## CLOSE\_PORT

The **CLOSE\_PORT** message is used to close a port on a Blue Heat/Net device.

### Message

*Message Number*

3 (0x03)

*Port Number*

Number of port in which to close.

*Data Length*

0 bytes

### Response

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>CLOSE_PORT</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

0 bytes

### Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

## See Also

[OPEN\\_PORT](#)

## EVENT\_OCCURRED

The **EVENT\_OCCURRED** message is used to report events to the host.

### Message

*Message Number*

34 (0x22)

**Port Number**

Number of the port in which the reported events apply to.

**Data Length**

19 bytes

**Data**

```
struct bhn_event_occurred
  uint16  events
  uint8   msr
  int32   oe_count
  int32   pe_count
  int32   fe_count
  int32   brk_count
```

**events**

The **events** member indicates which event types are being reported. Each event type has occurred if its corresponding bit is set. If a bit is cleared, the event type has not occurred. These bits are the same as the ones listed in the [SET\\_EVENT](#) message data.

**msr**

The current value of the Modem Status Register.

**oe\_count**

The current overrun error count.

**pe\_count**

The current parity error count.

**fe\_count**

The current framing error count.

**brk\_count**

The current break count.

**Response**

The **EVENT\_OCCURRED** message does not have a response.

**Remarks**

Events are reported via messages initiated from the Blue Heat/Net. These messages have no response, so the host need not reply to them.

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[SET\\_EVENT](#) [GET\\_EVENT](#)



## EVENT2\_OCCURRED

The **EVENT2\_OCCURRED** message is used to report events to the host.

### Message

*Message Number*

67 (0x43)

*Port Number*

Number of the port in which the reported events apply to.

*Data Length*

23 bytes

*Data*

```
struct bhn_event2_occurred
{
    uint16  events
    uint32  events2
    uint8   msr;
    int32   oe_count
    int32   pe_count
    int32   fe_count
    int32   brk_count
}
```

#### **events**

The **events** member indicates which event types are being reported. Each event type has occurred if its corresponding bit is set. If a bit is cleared, the event type has not occurred. These bits are the same as the ones listed in the **SET\_EVENT2** message data.

#### **events2**

The **events2** member indicates which event types are being reported. Each event type has occurred if its corresponding bit is set. If a bit is cleared, the event type has not occurred. These bits are the same as the ones listed in the **SET\_EVENT2** message data.

#### **msr**

The current value of the Modem Status Register.

#### **oe\_count**

The current overrun error count.

#### **pe\_count**

The current parity error count.

#### **fe\_count**

The current framing error count.

#### **brk\_count**

The current break count.

### Response

The **EVENT2\_OCCURRED** message does not have a response.

### Remarks

Events are reported via messages initiated from the Blue Heat/Net. These messages have no response, so the host need not reply to them.



## Requirements

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

## See Also

[SET\\_EVENT2](#) [GET\\_EVENT2](#)

## GET\_BHN\_INFO

The **GET\_BHN\_INFO** message is used to retrieve basic information about the Blue Heat/Net device.

### Message

*Message Number*

22 (0x16)

*Port Number*

N/A

*Data Length*

134 bytes

*Data*

```
struct bhn_info_cmd
  uint8  bhn_mac[6]
  uint8  bhn_hostname[128]
```

#### **bhn\_mac**

Optional MAC address to look for. See [Remarks](#) below for more details.

#### **bhn\_hostname**

Optional host name to look for. **bhn\_hostname** is a NULL-terminated string. See [Remarks](#) below for more details.

### Response

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_BHN_INFO</b> message succeeded.

*Data Length*

319 bytes

*Data*

```
struct bhn_info
  uint32  bhn_ser_num
  uint8   bhn_ip[4]
  uint8   reserved[16]
  uint8   bhn_mac[6]
  uint8   bhn_hostname[128]
  uint8   bhn_domainname[128]
  uint8   cds_ver[32]
  uint8   num_ports
```

**bhn\_ser\_num**

The serial number of the Blue Heat/Net unit.

**bhn\_ip**

The current runtime IPv4 address. **bhn\_ip** may differ from the IPv4 address stored in the flash configuration if the Blue Heat/Net is configured to use DHCP.

**bhn\_mac**

The MAC address of the Blue Heat/Net unit.

**bhn\_hostname**

The current runtime host name. **bhn\_hostname** may differ from the host name stored in the flash configuration if the Blue Heat/Net is configured to use DHCP.

**bhn\_hostname** is a NULL-terminated string.

**bhn\_domainname**

The current runtime domain name. **bhn\_domainname** may differ from the domain name stored in the flash configuration if the Blue Heat/Net is configured to use DHCP.

**bhn\_domainname** is a NULL-terminated string.

**cds\_ver**

The version of the flash *Configuration Data Space* (CDS) currently stored in the Blue Heat/Net unit.

**num\_ports**

The number of ports available on the Blue Heat/Net unit.

**Remarks**

A **GET\_BHN\_INFO** response is not always returned to the host, depending on the contents of the **GET\_BHN\_INFO** message. The response is only sent when at least one of following conditions is true:

- A. **struct bhn\_info\_cmd.bhn\_mac[0:5]** equals zero and **struct bhn\_info\_cmd.bhn\_hostname[0]** equals zero (a zero-length, NULL-terminated string).
- B. **struct bhn\_info\_cmd.bhn\_mac** equals the MAC address of the Blue Heat/Net unit.
- C. **struct bhn\_info\_cmd.bhn\_hostname** equals the current runtime host name of the Blue Heat/Net unit.

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	Yes
<b>Port message</b>	Yes
<b>Async</b>	N/A
<b>Sync</b>	N/A
<b>Port state must be</b>	N/A

**GET\_DATA\_MODE**

The **GET\_DATA\_MODE** message is used to retrieve a port's current data protocol mode.

**Message**

*Message Number*

51 (0x33)

*Port Number*

Number of the port in which to retrieve the data mode.

*Data Length*

0 bytes

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_DATA_MODE</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_DATA_MODE</b> message.

*Data Length*

1 byte

*Data*

```
struct bhn_data_mode
  uint8 mode
```

**mode**

The **mode** member specifies which data mode is currently set. **mode** can be one of the following values:

Value	Name	Description
0 (0x00)	BHN_DM_STREAM	Stream mode
1 (0x01)	BHN_DM_PACKET	Packet mode

See section 2.4, [Data Format](#), for details about stream and packet modes.

**Requirements**

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Any

**See Also**

[SET\\_DATA\\_MODE](#)

**GET\_EVENT**

The **GET\_EVENT** message is used to retrieve the current event settings (events last set by [SET\\_EVENT](#)).

**Message***Message Number*

33 (0x21)

*Port Number*

Number of the port in which to retrieve the current event settings.



*Data Length*

0 bytes

**Response**

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_EVENT</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_EVENT</b> message.

*Data Length*

2 bytes

*Data*

```
struct bhn_event
uint16  events
```

**events**

The **events** member indicates which event types are enabled or disabled. Each event type is enabled if its corresponding bit is set. If a bit is cleared, the event type is disabled. These bits are the same as the ones listed in the **SET\_EVENT** message data.

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[SET\\_EVENT EVENT OCCURRED](#)

## GET\_EVENT2

The **GET\_EVENT2** message is used to retrieve the current event settings (events last set by **SET\_EVENT2**).

**Message**

*Message Number*

66 (0x42)

*Port Number*

Number of the port in which to retrieve the current event settings.

*Data Length*

0 bytes

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_EVENT2</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>GET_EVENT2</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_EVENT2</b> message.

*Data Length*

6 bytes

*Data*

```
struct bhn_event2
  uint16  events
  uint32  events2
```

**events**

The **events** member indicates which event types are enabled or disabled. Each event type is enabled if its corresponding bit is set. If a bit is cleared, the event type is disabled. These bits are the same as the ones listed in the **SET\_EVENT2** message data.

**events2**

The **events2** member indicates which event types are enabled or disabled. Each event type is enabled if its corresponding bit is set. If a bit is cleared, the event type is disabled. These bits are the same as the ones listed in the **SET\_EVENT2** message data.

**Requirements**

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[SET\\_EVENT2 EVENT2 OCCURRED](#)

**GET\_FIFO\_CONTROL**

The **GET\_FIFO\_CONTROL** message is used to retrieve a port's FIFO control settings.

**Message***Message Number*

41 (0x29)

*Port Number*

Number of port in which to retrieve FIFO control settings.

*Data Length*

8 bytes

*Data*

The **GET\_FIFO\_CONTROL** message uses the same data structure as the **SET\_FIFO\_CONTROL** message, where the *mask* member is used to determine the requested data.

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_FIFO_CONTROL</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_FIFO_CONTROL</b> message.

*Data Length*

8 bytes

*Data*

The **GET\_FIFO\_CONTROL** response uses the same data as the **SET\_FIFO\_CONTROL** response, where the members contain the data requested by the **GET\_FIFO\_CONTROL** message's *mask* member.

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	No
<b>Port state must be</b>	Open

**See Also**

[SET\\_FIFO\\_CONTROL](#)

**GET\_LINE**

The **GET\_LINE** message is used to retrieve the line settings of the specified port.

**Message***Message Number*

9 (0x09)

*Port Number*

Number of port in which to retrieve the line settings.

*Data Length*

0 bytes

**Response****Version 9***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_LINE</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_LINE</b> message.

*Data Length*

26 bytes

*Data*

The **GET\_LINE** response uses the same data structure as the **SET\_LINE** message.

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[SET\\_LINE](#)

**GET\_LINE2**

The **GET\_LINE2** message is used to retrieve the line settings of the specified port.

**Message***Message Number*

63 (0x3F)

*Port Number*

Number of port in which to retrieve the line settings.

*Data Length*

0 bytes



## Response

### Status

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_LINE2</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>GET_LINE2</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_LINE2</b> message.

### Data Length

256 bytes

### Data

The **GET\_LINE2** response uses the same data structure as the **SET\_LINE2** message.

## Requirements

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

## See Also

[SET\\_LINE2](#)

## GET\_LINE\_STATE

The **GET\_LINE\_STATE** message is used to retrieve the current state of various modem signals.

## Message

### Message Number

12 (0x0C)

### Port Number

Number of port in which to retrieve line states.

### Data Length

0 bytes

## Response

### Status

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_LINE_STATE</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

### Data Length

1 byte

**Data**

```
struct bhn_lstate
  uint8  lines
```

**lines**

This member can be one or more of the following values:

Bit	Name	Description
0	LSTATE_CTS	CTS is active.
1	LSTATE_DSR	DSR is active.
2	LSTATE_RI	RI has gone from inactive to active.
3	LSTATE_DCD	DCD is active.
4	LSTATE_RTS	RTS is active.
5	LSTATE_DTR	DTR is active.

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**GET\_MSR**

The **GET\_MSR** message is used to retrieve the current state of the Modem Status Register.

**Message**

*Message Number*

36 (0x24)

*Port Number*

Number of the port in which to retrieve the Modem Status Register.

*Data Length*

0 bytes

**Response**

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_MSR</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

1 byte

*Data*

```
struct bhn_msr
  uint8  msr
```

**msr**

Current state of the Modem Status Register.



Bit	Description
0	Delta CTS.
1	Delta DSR.
2	Delta RI.
3	Delta CD.
4	CTS state.
5	DSR state.
6	RI state.
7	CD state.

### Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	No
<b>Port state must be</b>	Open

## GET\_PORTCAP

The **GET\_PORTCAP** message is used to retrieve the capabilities of the serial port hardware.

### Message

*Message Number*

39 (0x27)

*Port Number*

Number of the port in which to retrieve the capabilities. No matter which port number is specified, or if the message is sent to the general message port, **GET\_PORTCAP** always returns the capabilities of all ports.

*Data Length*

0 bytes

### Response

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_PORTCAP</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

16 byte

*Data*

```
struct bhn_msr
  int8  capability[16]
```

#### **capability**

An array of 16 characters, which indicates the capability of each port. The first character is for port #1, the second character is for port #2, and so on. Each character can be one of the following values:



Value	Description
'1' (0x31)	RS232 Only
'2' (0x32)	RS485(422) Only
'3' (0x33)	Switchable RS232/RS485(422)
'4' (0x34)	Multi-protocol (i.e. Blue Heat/Net Sync)

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	Yes
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Any

**GET\_PROTO\_SUPPORT**

The **GET\_PROTO\_SUPPORT** message is used to retrieve the protocol version(s) supported.

**Message**

*Message Number*

47 (0x2F)

*Port Number*

N/A

*Data Length*

0 bytes

**Response**

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_PROTO_SUPPORT</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.

*Data Length*

2 bytes

*Data*

```
struct bhn_proto_supp
  uint8  min
  uint8  max
```

**min**

Minimum protocol version supported.

**max**

Maximum protocol version supported.



## Requirements

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	N/A
<b>Sync</b>	N/A
<b>Port state must be</b>	N/A

## See Also

[GET\\_PROTO\\_VER](#)

## GET\_PROTO\_VER

The **GET\_PROTO\_VER** message is used to retrieve a string representation of the protocol version.

### Message

*Message Number*

46 (0x2E)

*Port Number*

N/A

*Data Length*

0 bytes

### Response

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_PROTO_VER</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

16 bytes

*Data*

```
struct bhn_proto_ver
int8 version[16]
```

#### **version**

NUL-terminated string containing the maximum protocol version supported. This string is formatted as "V<ver\_num>", where <ver\_num> is the maximum protocol version supported.

### Remarks

This message is deprecated and [GET\\_PROTO\\_SUPPORT](#) should be used instead.

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	Yes
<b>Port message</b>	Yes
<b>Async</b>	N/A
<b>Sync</b>	N/A
<b>Port state must be</b>	N/A

**See Also**[GET\\_PROTO\\_SUPPORT](#)**GET\_PSON**

The **GET\_PSON** message is used to retrieve the personality setting of a port.

**Message***Message Number*

42 (0x2A)

*Port Number*

Number of the port in which to retrieve the personality.

*Data Length*

0 bytes

**Response***Status*

<b>Status Value</b>	<b>Status Name</b>	<b>Description</b>
0 (0x00)	CMD_OK	The <b>GET_PSON</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

32 byte

*Data*

```
struct bhn_pson
  int8 personality[32]
```

**personality**

An array of 32 characters, which indicates the personality of the specified port. The personality setting is represented as a NUL-terminated string and can be one of the following values:



Value	Description
"ctid_portd"	Blue Heat/Net (this protocol)
"ppp-call"	PPP Call
"ppp-answer"	PPP Answer
"enserial"	Reserved
"rawsock"	Raw TCP server
"rawsockc"	Raw TCP client

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	Yes
<b>Port message</b>	No
<b>Async</b>	N/A
<b>Sync</b>	N/A
<b>Port state must be</b>	Any

**GET\_RTC\_ALARM**

The **GET\_RTC\_ALARM** message is used to retrieve the current alarm settings for the specified port.

**Message**

*Message Number*

56 (0x38)

*Port Number*

Number of port in which to retrieve the alarm settings.

*Data Length*

0 bytes

**Response**

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_RTC_ALARM</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>GET_RTC_ALARM</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_RTC_ALARM</b> message.

*Data Length*

6 bytes

*Data*

Same as **SET\_RTC\_ALARM** message data.

**Requirements**

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[SET\\_RTC\\_ALARM](#) [SET\\_RTC\\_CLOCK](#) [SET\\_RTC\\_MODE](#)

**GET\_RTC\_CLOCK**

The **GET\_RTC\_CLOCK** message is used to retrieve the current RTC time.

**Message**

*Message Number*

54 (0x36)

*Port Number*

Number of port in which to retrieve the RTC time.

*Data Length*

0 bytes

**Response**

*Status*

<b>Status Value</b>	<b>Status Name</b>	<b>Description</b>
0 (0x00)	CMD_OK	The <b>GET_RTC_CLOCK</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>GET_RTC_CLOCK</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_RTC_CLOCK</b> message.

*Data Length*

3 bytes

*Data*

Same as [SET\\_RTC\\_CLOCK](#) message data.

**Requirements**

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[SET\\_RTC\\_CLOCK](#) [SET\\_RTC\\_ALARM](#) [SET\\_RTC\\_MODE](#)



## GET\_RTC\_GATE

The **GET\_RTC\_GATE** message is used to retrieve the state of the transmitter/receiver gating control.

### Message

*Message Number*

58 (0x3A)

*Port Number*

Number of port in which to retrieve the gate state.

*Data Length*

0 bytes

### Response

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_RTC_GATE</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>GET_RTC_GATE</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_RTC_GATE</b> message.

*Data Length*

1 byte

*Data*

Same as **SET\_RTC\_GATE** message data.

### Requirements

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

### See Also

[SET\\_RTC\\_GATE](#) [SET\\_RTC\\_CLOCK](#)

## GET\_RTC\_MODE

The **GET\_RTC\_MODE** message is used to retrieve the current RTC mode settings.

### Message

*Message Number*

59 (0x3B)

*Port Number*

Number of port in which to retrieve the RTC mode.

*Data Length*

0 bytes

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>GET_RTC_MODE</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>GET_RTC_MODE</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>GET_RTC_MODE</b> message.

*Data Length*

1 byte

*Data*

```
struct bhn_rtc_mode
uint8 mode
```

**mode****mode** can be one or more of the following values:

Value	Name	Description
1 (0x01)	BHN_RTCM_MINUTE_RISING	Minutes pulse is on rising edge (otherwise falling edge).
2 (0x02)	BHN_RTCM_EXT_TIMEBASE	External timebase is selected (otherwise internal timebase).
4 (0x04)	BHN_RTCM_SEC_RISING	Seconds pulse is on rising edge (otherwise falling edge).
32 (0x20)	BHN_RTCM_MINUTE_PENDING	Indicates that the minute pulse once circuit is armed.
64 (0x40)	BHN_RTCM_MINUTE_STATE	Indicates that the state of the minute pulse signal is high ("on").

**Requirements**

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[SET\\_RTC\\_MODE](#) [SET\\_RTC\\_CLOCK](#) [SET\\_RTC\\_ALARM](#)

**OPEN\_PORT**

The **OPEN\_PORT** message is used to open a port on a Blue Heat/Net device. Most port-oriented messages require that the port be open before the message can be completed.

**Message***Message Number*

2 (0x02)



*Port Number*

Number of port in which to open.

*Data Length*

1 byte

*Data*

```
struct bhn_open
  uint8  flags
```

**flags**

This member can be one or more of the following values:

Bit	Name	Description
0	OPEN_READ	Read enabled.
1	OPEN_WRITE	Write enabled.

**Response**

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>OPEN_PORT</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

0 bytes

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Closed

**See Also**

[CLOSE\\_PORT](#)

**PING**

The **PING** message is a simple diagnostic message that can be used to test protocol communications.

**Message**

*Message Number*

1 (0x01)

*Port Number*

N/A

*Data Length*

0 bytes

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>PING</b> message succeeded.

*Data Length*

0 bytes

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	Yes
<b>Port message</b>	No
<b>Async</b>	N/A
<b>Sync</b>	N/A
<b>Port state must be</b>	N/A

**PURGE**

The **PURGE** message is used to remove any pending data in the receive or transmit streams.

**Message***Message Number*

7 (0x07)

*Port Number*

Number of port in which to purge.

*Data Length*

1 byte

*Data*

```
struct bhn_purge
  uint8  flags
```

**flags**

This member can be one or more of the following values:

Bit	Name	Description
0	PURGE_READ	Purge data coming into the Blue Heat/Net port.
1	PURGE_WRITE	Purge data going out of the Blue Heat/Net port.

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The PURGE message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

0 bytes



## Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

## RESET\_BHN

The **RESET\_BHN** message is used to reboot a Blue Heat/Net device.

### Message

*Message Number*

28 (0x1C)

*Port Number*

N/A

*Data Length*

0 bytes

### Response

*Status*

Status Value	Status Name	Description
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

0 bytes

### Remarks

The **RESET\_BHN** response cannot be depended on. It is possible that no response will be sent.

## Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	Yes
<b>Port message</b>	No
<b>Async</b>	N/A
<b>Sync</b>	N/A
<b>Port state must be</b>	N/A

## RESUME\_TX

The **RESUME\_TX** message is used to perform a software restart of a port that has been flow controlled by an XOFF. **RESUME\_TX** causes a fake XON to occur.

### Message

*Message Number*

19 (0x13)

*Port Number*

Number of port in which to resume transmission.



*Data Length*

0 bytes

### Response

The **RESUME\_TX** message has no response.

### Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	No
<b>Port state must be</b>	Open

### See Also

[SEND\\_XCHAR](#)

## SEND\_BREAK

The **SEND\_BREAK** message is used to transmit a break condition of a given length.

### Message

*Message Number*

13 (0x0D)

*Port Number*

Number of port in which to transmit a break condition.

*Data Length*

1 byte

*Data*

```
struct bhn_break
  uint8  duration
```

#### **duration**

If **duration** is zero, **SEND\_BREAK** transmits a break for at least 0.25 seconds. If **duration** is not zero, **SEND\_BREAK** transmits a break for at least **duration**\*0.10 seconds (in other words, **duration** is the number of 10ths of a second).

### Response

The **SEND\_BREAK** message has no response.

### Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	No
<b>Port state must be</b>	Open

### See Also

[START\\_STOP\\_BREAK](#)



## SEND\_IMMEDIATE

The **SEND\_IMMEDIATE** message is used to transmit a character outside of the normal data stream.

### Message

*Message Number*

30 (0x1E)

*Port Number*

Number of port in which to transmit a character.

*Data Length*

1 byte

*Data*

```
struct bhn_sendi
uint8  chr
```

***chr***

Character to transmit.

### Response

The **SEND\_IMMEDIATE** message has no response.

### Remarks

Depending on the architecture of the port device, a character transmitted using **SEND\_IMMEDIATE** may have to wait until the characters in the FIFO have been sent.

### Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	No
<b>Port state must be</b>	Open

## SEND\_XCHAR

The **SEND\_XCHAR** message is used to send a software flow control character.

### Message

*Message Number*

37 (0x25)

*Port Number*

Number of the port in which to send a software flow control character.

*Data Length*

1 byte

*Data*

```
struct bhn_xchar
uint8  xchar
```

***xchar***

Flow control character to send. The ***xchar*** member can be one of the following values:



Value	Name	Description
1 (0x01)	XCHAR_XON	Send a START character.
2 (0x02)	XCHAR_XOFF	Send a STOP character.

The actual characters sent when using **SEND\_XCHAR** are configured using the **SET\_LINE** message.

## Response

### Status

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SEND_XCHAR</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SEND_XCHAR</b> message.

### Data Length

0 bytes

## Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	No
<b>Port state must be</b>	Open

## See Also

[SET\\_LINE RESUME TX](#)

## SET\_DATA\_MODE

The **SET\_DATA\_MODE** message is used to change a port's data protocol mode between stream and packet modes.

## Message

### Message Number

50 (0x32)

### Port Number

Number of the port in which to change the data mode.

### Data Length

1 byte

### Data

```
struct bhn_data_mode
  uint8  mode
```

### mode

The **mode** member specifies which data mode to set. **mode** can be one of the following values:



Value	Name	Description
0 (0x00)	BHN_DM_STREAM	Stream
1 (0x01)	BHN_DM_PACKET	Packet

See section 2.4, [Data Format](#), for details about stream and packets modes.

## Response

### Status

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_DATA_MODE</b> message succeeded.
1 (0x01)	BHN_ERROR_OPENED	The port is currently open, but <b>SET_DATA_MODE</b> requires it to be closed.
9 (0x09)	BHN_ERROR_INVALID_PARM	The specified data mode is not valid.
10 (0x0A)	BHN_ERROR_UNSUPPORTED	The specified data mode is not supported on the specified port.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SET_DATA_MODE</b> message.

### Data Length

0 bytes

## Requirements

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Closed

## See Also

[GET\\_DATA\\_MODE](#)

## SET\_DTR

The **SET\_DTR** message is used to set the state of the DTR signal.

### Message

#### Message Number

16 (0x10)

#### Port Number

Number of port in which to set the state of DTR.

#### Data Length

1 byte

#### Data

```
struct bhn_rtsdtr
  uint8  onoff
```

#### **onoff**

If **onoff** is non-zero, **SET\_DTR** sets the DTR signal to a logical 1 (on). If **onoff** is zero, **SET\_DTR** sets the DTR signal to a logical 0 (off).

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The SET_DTR message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

0 bytes

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**[SET\\_RTS\\_DTR SET\\_RTS](#)**SET\_EVENT**

The **SET\_EVENT** message is used to request event reporting. When the data mode (**SET\_DATA\_MODE**) is set to **BHN\_DM\_STREAM**, events enabled by **SET\_EVENT** are reported via **EVENT\_OCCURRED** messages. If however the data mode is set to **BHN\_DM\_PACKET**, the events are reported via **EVENT2** directors.

**Message***Message Number*

32 (0x20)

*Port Number*

Number of the port in which to request event reporting.

*Data Length*

2 bytes

*Data*

```
struct bhn_event
  uint16 events
```

**events**

The **events** member indicates which event types are to be enabled or disabled. Each event type is enabled by setting its corresponding bit, as shown below. Clearing a bit disables the event type.



Bit	Name	Description
0	EVENT_OE	Overrun error events.
1	EVENT_PE	Parity error events.
2	EVENT_FE	Framing error events.
3	EVENT_BREAK	Break events.
4	EVENT_CTS	CTS change of state events.
5	EVENT_DSR	DSR change of state events.
6	EVENT_RING	RING events.
7	EVENT_DCD	DCD change of state events.
8	EVENT_RX_XON	Start character received events.
9	EVENT_RX_XOFF	Stop character received events.
10-15	Reserved	
N/A	EVENT_NONE	Disables all events.
N/A	EVENT_ALL	Enables all events.

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The SET_EVENT message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the SET_EVENT message.

*Data Length*

0 bytes

**Remarks**

Events that are reported via **EVENT\_OCCURRED** messages, initiated from the Blue Heat/Net, have no response. Therefore, the host need not reply to them.

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[GET\\_EVENT SET\\_DATA MODE\\_EVENT\\_OCCURRED EVENT2](#)

**SET\_EVENT2**

The **SET\_EVENT2** message is used to request event reporting. When the data mode (**SET\_DATA\_MODE**) is set to **BHN\_DM\_STREAM**, events enabled by **SET\_EVENT2** are reported via **EVENT\_OCCURRED** messages. If however the data mode is set to **BHN\_DM\_PACKET**, the events are reported via **EVENT2** directors.

**Message**

*Message Number*

65 (0x41)

**Port Number**

Number of the port in which to request event reporting.

**Data Length**

6 bytes

**Data**

```
struct bhn_event2
  uint16  events
  uint32  events2
```

**events**

The **events** member indicates which event types are to be enabled or disabled. Each event type is enabled by setting its corresponding bit, as shown below. Clearing a bit disables the event type.

Bit	Name	Description
0	EVENT_OE	Overrun error events.
1	EVENT_PE	Parity error events.
2	EVENT_FE	Framing error events.
3	EVENT_BREAK	Break events.
4	EVENT_CTS	CTS change of state events.
5	EVENT_DSR	DSR change of state events.
6	EVENT_RING	RING events.
7	EVENT_DCD	DCD change of state events.
8	EVENT_RX_XON	Start character received events.
9	EVENT_RX_XOFF	Stop character received events.
10-15	Reserved	
N/A	EVENT_NONE	Disables all events.
N/A	EVENT_ALL	Enables all events.

**events2**

The **events2** member indicates which event types are to be enabled or disabled. Each event type is enabled by setting its corresponding bit, as shown below. Clearing a bit disables the event type.



Bit	Name	Description
0	EVENT2_IDLE_RECVD	
1	EVENT2_ABORT1	HDLC/SDLC Abort or Go Ahead sequence.
2	EVENT2_RCC_UNDER	
3	EVENT2_PRE_SENT	
4	EVENT2_IDLE_SENT	
5	EVENT2_ABORT_SENT	
6	EVENT2_EOFM_SENT	
7	EVENT2_CRC_SENT	
8	EVENT2_TX_UNDER	
9	EVENT2_RI_FALL	
10	EVENT2_RI_RISE	
11	EVENT2_DCD_FALL	
12	EVENT2_DCD_RISE	
13	EVENT2_CTS_FALL	
14	EVENT2_CTS_RISE	
15	EVENT2_DSR_FALL	
16	EVENT2_DSR_RISE	
17	EVENT2_DPLL_DSYNC	
18	EVENT2_ABORT2	HDLC/SDLC with QAbort set, occurs for character followed by an Abort sequence.
19	EVENT2_CRC	
20	EVENT2_SHORT_FR_CV	
21	EVENT2_RX_BUF_LOW	
22	EVENT2_TX_BUF_LOW	
23	EVENT2_RX_BUF_HIGH	
24	EVENT2_TX_BUF_HIGH	
25-31	Reserved	
N/A	EVENT2_NONE	Disable all events.
N/A	EVENT2_ALL	Enable all events.

## Response

### Status

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_EVENT2</b> message succeeded.
9 (0x09)	BHN_ERROR_INVALID_PARM	An error occurred setting the specified event masks.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>SET_EVENT2</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SET_EVENT2</b> message.

### Data Length

0 bytes

## Remarks

Events that are reported via **EVENT\_OCCURRED** messages, initiated from the Blue Heat/Net, have no response. Therefore, the host need not reply to them.



## Requirements

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

## See Also

[GET\\_EVENT2 SET\\_DATA\\_MODE\\_EVENT2 OCCURRED\\_EVENT2](#)

## SET\_FIFO\_CONTROL

The **SET\_FIFO\_CONTROL** message is used to configure a port's FIFO control settings.

### Message

*Message Number*

31 (0x1F)

*Port Number*

Number of port in which to set the FIFO control settings.

*Data Length*

8 bytes

*Data*

```
struct bhn_fifo
  uint8  mask
  uint16 fifo_size
  uint8  tx_threshold
  uint8  rx_threshold
  uint16 tx_load_size
  uint8  flags
```

### **mask**

The **mask** member specifies which of the other members contain valid data. When this structure is used with the **GET\_FIFO\_CONTROL** message, the **mask** member indicates which other members to retrieve. This member can be one or more of the following values:

Value	Name	Description
1 (0x01)	FIFO_MASK_FIFO_SIZE	<i>fifo_size</i> is valid
2 (0x02)	FIFO_MASK_TX_THRESHOLD	<i>tx_threshold</i> is valid
4 (0x04)	FIFO_MASK_RX_THRESHOLD	<i>rx_threshold</i> is valid
8 (0x08)	FIFO_MASK_TX_LOAD_SIZE	<i>tx_load_size</i> is valid
16 (0x10)	FIFO_MASK_FLAGS	<i>flags</i> is valid
31 (0x1F)	FIFO_MASK_ALL	All members are valid

### **fifo\_size**

The size of the UART FIFOs. This member is ignored during the **SET\_FIFO\_CONTROL** message.

### **tx\_threshold**

Flag that indicates the transmit FIFO threshold setting. This value is expressed as a mode instead of an actual value. This member can be one of the following values:



Value	Name	Description
0 (0x00)	FIFO_TT_MODE1	least CPU usage (transmit threshold set low)
1 (0x01)	FIFO_TT_MODE2	balanced (transmit threshold set in the middle)
2 (0x02)	FIFO_TT_MODE3	best throughput (transmit threshold set high)

### ***rx\_threshold***

Flag that indicates the receive FIFO threshold setting. This value is expressed as a mode instead of an actual value. This member can be one of the following values:

Value	Name	Description
0 (0x00)	FIFO_RT_MODE1	least latency (receive threshold set low)
1 (0x01)	FIFO_RT_MODE2	balanced (receive threshold set in the middle)
2 (0x02)	FIFO_RT_MODE3	best throughput (receive threshold set high)

### ***tx\_load\_size***

The amount of transmit data placed into the transmit FIFO, once the previous transmit data has been sent. This value can be between one (1) and ***fifo\_size*** (inclusive).

### ***flags***

Flags that indicate other control options. This member can be one or more of the following values:

Value	Name	Description
1 (0x01)	FIFO_FLAG_FIFOS_ENABLED	FIFO buffers are enabled. If this flag is not specified, FIFOs are disabled and the <b><i>tx_threshold</i></b> , <b><i>rx_threshold</i></b> , and <b><i>tx_load_size</i></b> members have no effect (note; their values can still be set, but only take effect when FIFOs are enabled).

## Response

### *Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_FIFO_CONTROL</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
8 (0x08)	BHN_ERROR_INVALID_FIFO	One or more FIFO settings are invalid. The mask member of the response data specifies which members were invalid. Settings are applied in an all-or-none manner. Therefore, this response status means that no settings were applied.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SET_FIFO_CONTROL</b> message.

### *Data Length*

8 bytes

### *Data*

The response data is in the form of a **bhn\_fifo** structure.

If the response status is **BHN\_ERROR\_INVALID\_FIFO**, the response data's **mask** member indicates which values were invalid. The invalid values are also provided in the other structure members. See the message data section for a list of valid flags and values.

If the response status is not **BHN\_ERROR\_INVALID\_FIFO**, the response Data should be treated as undefined and not used in any way.



## Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	No
<b>Port state must be</b>	Open

## See Also

[GET\\_FIFO\\_CONTROL](#)

## SET\_LINE

The **SET\_LINE** message is used to change the line settings of the specified port.

### Message

*Message Number*

8 (0x08)

*Port Number*

Number of port in which to change the line settings.

*Data Length*

26 bytes

*Data*

```
struct bhn_lset
  uint32  baud
  uint8   databits
  uint8   parity
  uint8   stopbits
  uint8   sflow
  uint8   xoff
  uint8   xon
  uint8   hflow
  uint8   lloop
  uint8   special_char_mode
  uint8   error_char
  uint8   break_char
  uint8   event_char
  uint8   escape_char
  uint8   use_x_lim
  uint32  xoff_lim
  uint32  xon_lim
```

### ***baud***

To be completed.

### ***databits***

To be completed.

### ***parity***

To be completed.

### ***stopbits***

To be completed.

### ***sflow***

To be completed.

**xoff**

To be completed.

**xon**

To be completed.

**hflow**

To be completed.

**lloop**

To be completed.

**special\_char\_mode**

To be completed.

**error\_char**

To be completed.

**break\_char**

To be completed.

**escape\_char**

To be completed.

**use\_x\_lim**

To be completed.

**xoff\_lim**

To be completed.

**xon\_lim**

To be completed.

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_LINE</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
4 (0x04)	BHN_ERROR_INVALID_BAUD	An unsupported baud rate was specified.
6 (0x06)	BHN_ERROR_INVALID_PARITY	An invalid parity mode was specified.
7 (0x07)	BHN_ERROR_INVALID_DATA_BITS	An invalid number of data bits was specified.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SET_LINE</b> message.

*Data Length*

0 bytes

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**[GET\\_LINE](#)



## SET\_LINE2

The **SET\_LINE2** message is used to change the line settings of the specified port.

### Message

*Message Number*

62 (0x3E)

*Port Number*

Number of port in which to change the line settings.

*Data Length*

256 bytes

### Data

```
struct bhn_lset2
  uint8 reserved

  struct
  | uint8 tx
  | uint8 rx
  smode

  struct
  | uint8 interrupt_threshold
  | uint8 reserved[31]
  misc_receiver

  struct
  | uint8 interrupt_threshold
  | uint8 reserved[31]
  misc_transmitter

  struct
  | uint32 ref_freq
  | uint32 bps_error
  | uint32 bps
  | uint8 bps_frac
  | uint8 async_div
  | uint8 dpll_div
  | uint8 ctr_div
  | uint8 enc_dec
  |
  | struct
  | | uint8 A
  | | uint8 B
  | | uint8 C
  | | uint8 D
  | clk_tree
  |
  | uint8 clk_pin
  rxclk
```



```
struct
| uint32 ref_freq
| uint32 bps_error
| uint32 bps
| uint8 bps_frac
| uint8 async_div
| uint8 dpll_div
| uint8 ctr_div
| uint8 enc_dec
|
| struct
| | uint8 A
| | uint8 B
| | uint8 C
| | uint8 D
| clk_tree
|
| uint8 clk_pin
txclk

struct
| uint8 pre_pat
| uint8 pre_len
| uint8 tx_idle
tx_pre_idle

uint8 line_mode
uint8 duplex_mode

struct
| uint8 tx0
| uint8 tx1
| uint8 tx_len
| uint8 rx0
| uint8 rx1
| uint8 rx_len
| uint8 strip_sync
sync_addr

struct
| uint8 tx
| uint8 rx
dbits

uint8 tx_frac_stop

struct
| uint8 tx
| uint8 rx
parity

struct
| uint8 tx_type
| uint8 tx_start
| uint8 tx_mode
| uint8 rx_type
| uint8 rx_start
crc

uint8 tx_trig_gate
uint8 rx_gate

uint8 reserved[121]
```

**smode**

Establishes the operational serial mode for the specified port. There are 2 sub-settings in this item, a setting for the Transmitter and a setting for the Receiver. Although the Blue Heat/Net allows the settings for the Receiver and Transmitter of a given port to be different, in many cases different settings are incompatible. Depending on the clocking resources selected (via the **txclk** and **rxclk** members) there are certain serial mode combinations that can be used together on a given port. **smode.tx** and **smode.rx** can be one of the following values:

Value	Name	Description
0 (0x00)	SMODE_ASYNC	Asynchronous.
1 (0x01)	SMODE_EXT_SYNC	External Sync (Receiver only).
2 (0x02)	SMODE_ISO	Isochronous.
4 (0x04)	SMODE_MONO	MonoSync.
5 (0x05)	SMODE_BISYNC	BiSync.
6 (0x06)	SMODE_HDLC	HDLC/SDLC.
7 (0x07)	SMODE_TRANS_BI	Transparent BiSync.
8 (0x08)	SMODE_9BIT	9-Bit Asynchronous.

**misc\_receiver**

Contains a collection of miscellaneous receiver settings.

**interrupt\_threshold**

Sets the receiver interrupt threshold. Valid values are 1 to 32, inclusive.

**misc\_transmitter**

Contains a collection of miscellaneous transmitter settings.

**interrupt\_threshold**

Sets the transmitter interrupt threshold. Valid values are 1 to 32, inclusive.

**rxclk/txclk****ref\_freq**

This number is the frequency, in Hz, of the reference that is used to generate the bit rate, when that signal source is routed through either of the Baud Rate Generators. There are 4 possible reference frequency sources:

- The Internal reference clock (18.432 MHz).
- An External reference clock (A signal applied through an Input pin).
- The RxC Line Interface signal (which can be either an Input or an Output).
- The TxC Line Interface signal (which can be either an Input or an Output).

See **clk\_tree** settings below, for details about clocking.

Valid values for **ref\_freq** are from 0 to 20,000,000, inclusive. If the internal reference clock is used, **INT\_REF\_FREQ** can be used for the value of **ref\_freq**.

**bps\_error**

The maximum allowable bit rate error in PPM (parts per million), 10000 would equal 1%.

With the IUSC device there are multiple ways to achieve a given bit rate. To make the best use of the internal clocking resources of the IUSC device, the bit rate computational algorithm uses this **bps\_error** value to decide when a given computation is "good enough". Set this value to something realistic for the application. If this value is always set to zero, the algorithm will likely to return



an error code, unless the desired bit rate can be achieved exactly (with zero error).

**bps**

Bit rate integer portion.

A value from 0 to N (when N can be any value up to the **ref\_freq** value).

**bps\_frac**

Bit rate fractional portion (in units of 10th's).

Only effective at low bit rates. As the bit rate goes down the setting resolution increases. To get an approximation of the bps resolution over a range of bit rates, perform the following calculation:

$$bps\_resolution = \frac{bps2 - bps1}{\left(\frac{ref}{bps1}\right) - \left(\frac{ref}{bps2}\right)}$$

*bps1* = lower data rate in bits per second

*bps2* = upper data rate in bits per second

*ref* = clock source frequency

Example:

$$bps1 = 9000$$

$$bps2 = 10000$$

$$ref = 18.432MHz$$

$$\begin{aligned} bps\_resolution &= \frac{10000 - 9000}{\left(\frac{18432000}{9000}\right) - \left(\frac{18432000}{10000}\right)} \\ &= \frac{1000}{2048 - 1843} \\ &= \frac{1000}{205} \\ &= 4.88 \end{aligned}$$

Note: Do not do the calculation over too small or too large a range, or the results will be misleading.

**bps** and **bps\_frac** are combined to form a bit rate number that is used in the "Bit Rate Computational Algorithm"

**async\_div**

Sets the fundamental clock division setting for the Receiver or Transmitter. Only settings 16, 32 or 64 are allowed and are only relevant to Async mode of operation.

**dpll\_div**

Sets the fundamental clock division setting for the DPLL Receiver clock recovery circuit within the IUSC device. Only settings 8, 16 or 32 are allowed and are only relevant to modes which enable the use of the DPLL as the clocking source for the Receiver (see **clk\_tree** settings).

### **ctr\_div**

Sets the divisor used for the Counter selected to clock the Receiver or Transmitter. Only settings 4, 8, 16 or 32 are allowed and are only required when a Counter is setup to directly clock the Receiver, Transmitter, RxC or TxC signals (see **clk\_tree** settings).

### **enc\_dec**

This selects the Receiver data decoding or Transmitter data encoding method to be employed. **enc\_dec** can be one of the following values:

Value	Name	Description
0 (0x00)	NRZ	NRZ.
1 (0x01)	NRZ_INV	NRZB.
2 (0x02)	NRZI_MARK	NRZI-Mark.
3 (0x03)	NRZI_SPACE	NRZI-Space.
4 (0x04)	BIPH_MARK	Biphase-Mark.
5 (0x05)	BIPH_SPACE	Biphase-Space.
6 (0x06)	BIPH_LEVEL	Biphase-Level.
7 (0x07)	BIPH_DIFF	Differential Biphase-Level.

Only the **NRZ** and **NRZ\_INV** settings are valid when the **smode** setting is set to **SMODE\_ASYNC**.

All **NRZI\_Xxx** and **BIPH\_Xxx** settings expect that the Receiver is setup to be clocked from the DPLL (see **clk\_tree** settings).

### **clk\_tree**

Refer to [Figure 3-1](#), which shows the internal clocking structure of the IUSC device and how it relates to the settings described below. Having a good understanding of the clock tree settings is important to achieve a successful setup. The clock tree settings (along with bit rate parameter settings) allow many possible combinations to be selected, but many of those combinations are inconsistent or incompatible. In fact, there are many more combinations that will not work as opposed to combinations that will work. Here are some examples of **incompatible** combinations.

Simple case: If the Receiver and Transmitter were both setup to be clocked from Baud Rate Generator-0 (**clk\_tree.A = CLK\_BRGO**), and the bit rate parameters for the Receiver are different than the similar settings for the Transmitter, then an error would result.

More complicated case: If the Receiver was setup to be clocked by the RxC signal input (**rxclk.clk\_tree.A = CLK\_RXC**), and the RxC signal was setup to be an output sourced from the Receiver (**rxclk.clk\_pin = RXC\_RXCLK**), this creates a "circular" clocking combination which will not operate. An error code will result.

Somewhat obscure case: If the Receiver was setup with a very low **bps\_error** with a "non-standard" bit rate and/or a high bit rate ( $\geq 1.8\text{M bps}$ ) using Baud Rate Generator-0, and the Transmitter was setup with a low bit rate ( $\leq 280\text{ bps}$ ) using Baud Rate Generator-1. This combination would cause an error because the internal "Counter" resources of the IUSC device can not support this diverse range of settings.

There are four sub-members of the **clk\_tree** member, each relates to a different "level" as shown in [Figure 3-1](#). This figure shows the selections available at the different levels, using different colours for better understanding.

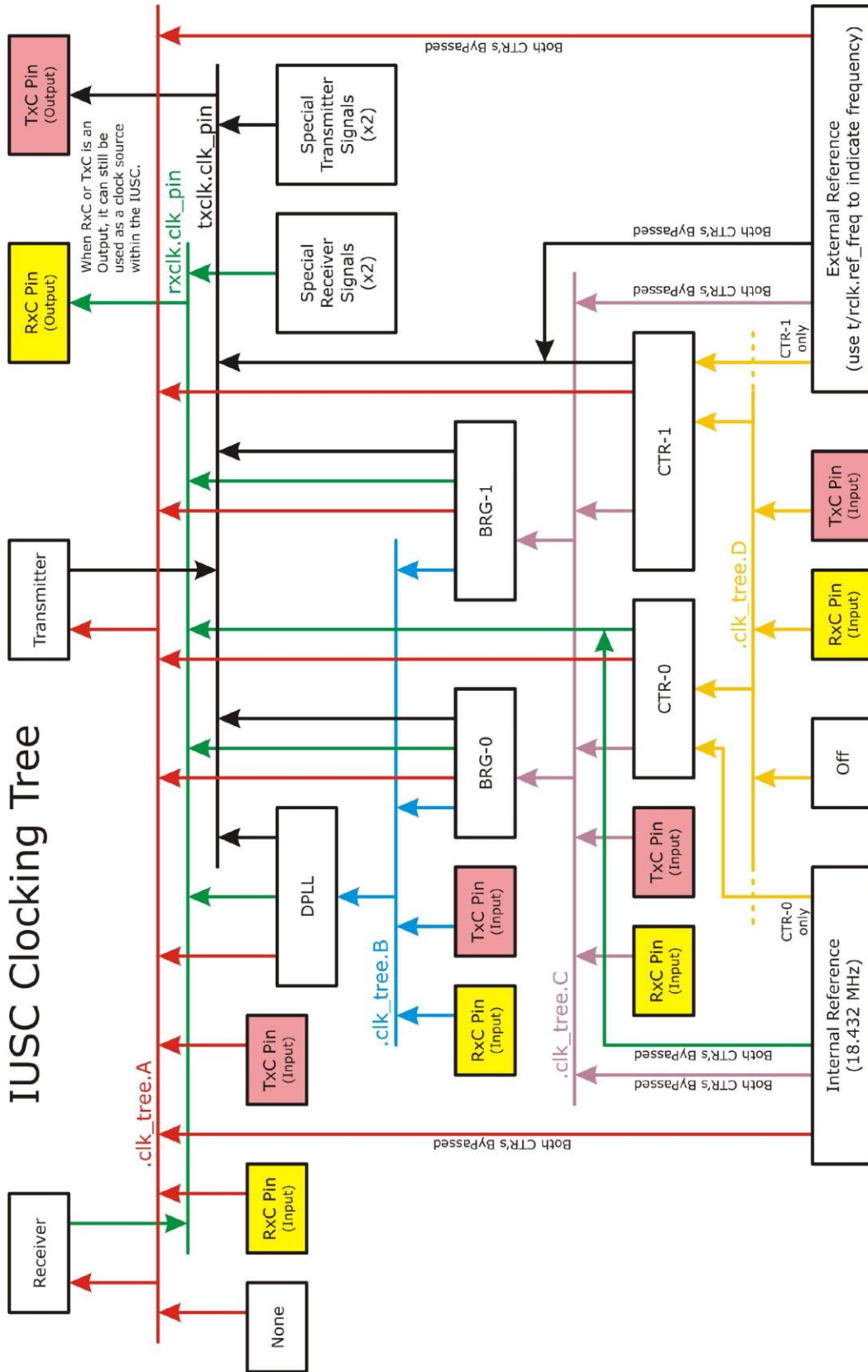


Figure 3-1

CTR-0 and CTR-1 are "By-passed" either by an explicit selection at a particular "clk\_tree" level, or by a bit rate computation which yields a divisor which excludes the use of a Counter.

The counters are by-passed together (because there is only one bit in the IUSC that controls the counter by-pass selection).

**Bit Rate Computational Algorithm**

This algorithm examines the entire clocking tree setup and the related bit rate parameters (*ref\_freq*, *bps*, *bps\_frac* and *bps\_error*) to attempt to find a solution that achieves the desired bit rate within the specified error (*bps\_error*). If no solution can be found the settings are rejected and an error code is returned. The algorithm gives priority to the setup of the various sections of the IUSC device in the following order; first the Receiver, then Transmitter, then RxC output clock signal, then TxC output clock signal. The algorithm takes into account previous uses of clocking resource settings during the algorithm; this is how it detects combinations that cannot be achieved, or which are incompatible.

**A**

**A** selects the primary clock source for either the Receiver or Transmitter and can be one of the following values:

Value	Name	Description
0 (0x00)	CLK_NONE	No clock applied (turns off the respective Serial section).
1 (0x01)	CLK_RXC	From RxC External Line Interface signal.
2 (0x02)	CLK_TXC	From TxC External Line Interface signal.
3 (0x03)	CLK_DPLL	From DPLL.
4 (0x04)	CLK_BRG0	From Baud rate generator 0.
5 (0x05)	CLK_BRG1	From Baud rate generator 1.
6 (0x06)	CLK_CTR0	From CTR0 (by-passes BRG).
7 (0x07)	CLK_CTR1	From CTR1 (by-passes BRG).
134 (0x86)	CLK_INT_REF	From Internal reference (18.432 MHz).
135 (0x87)	CLK_EXT_REF	From External reference (External Reference Signal input).

To use the **CLK\_DPLL** setting to clock the Transmitter, the Receiver must also be set to **CLK\_DPLL**, otherwise an error is returned.

The **CLK\_NONE** selection effectively disables the section to which it is applied, and could be used if you want a port to be a Transmit or Receive only port.

Some special consideration should be taken when selecting **CLK\_INT\_REF** or **CLK\_EXT\_REF**. While these clocking sources are physically possible, they come at a high "cost" to the clocking structure. This occurs because the IUSC device uses only one bit to control the bypassing of CTR-0 and CTR-1. See section 4.3 of the IUSC manual and [Figure 3-1](#).

**B**

This selects the clock source for the Receiver DPLL, and is only required when the *clk\_tree.A* selection is **CLK\_DPLL**. When this selection is made, the *dpll\_div* value must be set to an appropriate value. If not needed, **B** can be zero, but must be one of the following values:

Value	Name	Description
0 (0x00)	DPLL_BRG0	From BRG0.
1 (0x01)	DPLL_BRG1	From BRG1.
2 (0x02)	DPLL_RXC	From RxC External Line Interface signal.
3 (0x03)	DPLL_TXC	From TxC External Line Interface signal.

**C**

This selects the clock source for the Baud Rate Generator, and is only



required when either the *clk\_tree.A* setting is **CLK\_BRG0** or **CLK\_BRG1** or the *clk\_tree.B* setting is **DPLL\_BRG0** or **DPLL\_BRG1**. The selection made at either *clk\_tree.A* or *clk\_tree.B* determines which Baud Rate Generator is setup. *B* can be zero if not required, but must be one of the following values:

Value	Name	Description
0 (0x00)	BRG_CTR0	From IUSC CTR-0 (or Port-0 if INT_REF is set).
1 (0x01)	BRG_CTR1	From IUSC CTR-1 (or Port-1 if EXT_REF is set).
2 (0x02)	BRG_RXC	From RxC external line interface signal.
3 (0x03)	BRG_TXC	From TxC external line interface signal.
128 (0x80)	BRG_INT_REF	From Internal reference (18.432 MHz).
129 (0x81)	BRG_EXT_REF	From External reference (External Reference Signal input).

Some special consideration should be taken when selecting **BRG\_INT\_REF** or **BRG\_EXT\_REF**. While these clocking sources are physically possible, they come at a high "cost" to the clocking structure. This occurs because the IUSC device uses only one bit to control the bypassing of CTR-0 and CTR-1. See section 4.3 of the IUSC manual and [Figure 3-1](#).

#### **D**

This establishes the clocking source for either CTR-0 or CTR-1. The selection of which counter to setup is made at either *clk\_tree.A* or *clk\_tree.C*. *D* can be one of the following values:

Value	Name	Description
0 (0x00)	CTR_DISABLE	Disable counter.
1 (0x01)	CTR_REF	From their respective references: CTR-0 from Internal reference (18.432 MHz), CTR-1 is clocked from External reference.
2 (0x02)	CTR_RXC	From RxC external line interface signal.
3 (0x03)	CTR_TXC	From TxC external line interface signal.

#### **clk\_pin**

This setting establishes two operational aspects of the RxC and TxC line interface signals.

- Whether the signal is an Input or an Output
- When an Output, what source is connected to the signal.

These signals are part of the interface signals available on the 25D I/O connector. These signals are commonly used as data clocks when operating in various Synchronous modes. Although they are named RxC (Receiver Clock) and TxC (Transmitter Clock), that naming is for identification purposes only, either signal can be an output or an input and either can clock the Receiver or Transmitter. Circuitry of the Blue Heat/Net automatically detects when these signals are selected to be inputs or outputs and switches the "direction" of the Line Interface circuits accordingly.

The setting for the RxC signal is found in *rxclk.clk\_pin*, and for the TxC signal in *txclk.clk\_pin*. *clk\_pin* can be one of the following values:



Value	Name	Description
0 (0x00)	XC_INP	Input.
1 (0x01)	TXC_TXCLK	Output, mirror the Transmitter clock (TxC pin only).
1 (0x01)	RXC_RXCLK	Output, mirror the Receiver clock (RxC pin only).
2 (0x02)	XC_CHAR_CLK	Output, RxCHAR for RxC, TxCHAR for TxC.
3 (0x03)	TXC_TXCOMP	Output, TX Complete (TxC pin only).
3 (0x03)	RXC_RXSYNC	Output, RxSYNC (RxC pin only).
4 (0x04)	XC_BRG0	Output, from BRG-0.
5 (0x05)	XC_BRG1	Output, from BRG-1.
6 (0x06)	TXC_CTR1	Output, from CTR-1 (TxC pin only).
6 (0x06)	RXC_CTR0	Output, from CTR-0 (RxC pin only).
134 (0x86)	TXC_PORT1	Output, from Port-1 (TxC pin only).
134 (0x86)	RXC_PORT0	Output, from Port-0 (RxC pin only).
7 (0x07)	XC_DPLL	Output, from DPLL-RX for RxC, DPLL-TX for TxC.

One interesting point here, is that when either the RxC or TxC signal is used as a clocking source at the appropriate levels of the clock tree, the RxC or TxC signals do NOT have to be setup as inputs (as would seem intuitive). The RxC or TxC signals can be outputs as well as being clock sources for various items, the IUSC device takes care of the signal routing internally.

When either the RxC or TxC signal is setup to be sourced from a Baud Rate generator or Counter, then the settings defined in **ref\_freq**, **bps**, **bps\_frac**, **bps\_error** or **ctr\_div** must be appropriately setup. For the RxC signal, the settings are placed in the **rxclk** member, and for the TxC signal in the **txclk** member.

### **tx\_pre\_idle**

#### **pre\_pat**

Sets the Preamble pattern. A pattern that is transmitted for N number of bits at the beginning of each Frame (or Packet), usually used with "Character-oriented Synchronous" protocols (MonoSync, BiSync, or Transparent BiSync) when the data includes encoded clock information (NRZI or BP-Phase clock encoding). A Preamble pattern is generally used to prepare the Receiver DPLL for subsequent data (ie: gets the Receiver "sync'd" to the bit transition point of the Transmitter).

**pre\_pat** can be one of the following values:

Value	Name	Description
0 (0x00)	PREAM_DISABLE	Disabled.
1 (0x01)	PREAM_ZEROS	All zero's.
2 (0x02)	PREAM_ONES	All one's.
3 (0x03)	PREAM_FLAGS	Flags (only in HDLC mode).
4 (0x04)	PREAM_ZERO_ONE	0101... pattern.
5 (0x05)	PREAM_ONE_ZERO	1010... pattern.

#### **pre\_len**

Establishes the length of the Preamble pattern as 8, 16, 32, or 64 bits long.

#### **tx\_idle**

This sets the pattern that is transmitted when the Transmitter goes "Idle". Idle occurs after any CRC and/or closing SYNC (or Flag) that may be setup for the protocol (setup by the **smode** member).



Value	Name	Description
0 (0x00)	IDLE_DEF	Default idle activity for a given serial mode.
1 (0x01)	IDLE_ALT01	Alternating zero's and one's (encoded).
2 (0x02)	IDLE_ZEROS	Continuous zero's (encoded).
3 (0x03)	IDLE_ONES	Continuous one's (encoded).
5 (0x05)	IDLE_MRK_SPC	Alternating Mark (one) and Space (zero).
6 (0x06)	IDLE_SPACE	Continuous zero's (NOT encoded).
7 (0x07)	IDLE_MARK	Continuous ones's (NOT encoded).

### ***line\_mode***

This setting establishes the electrical interface for the signals that connect to the 25D I/O connector. The Interface is implemented with an SP508 device which allows the selection of electrical interfaces common to Synchronous communications protocols. *line\_mode* can be one of the following values:

Value	Name	Description
1 (0x01)	LM_530A	RS422 (with V.10 DTR).
2 (0x02)	LM_530	RS422 (all signals V.11).
3 (0x03)	LM_X21	Identical to LM_530 (except RI not available).
4 (0x04)	LM_V35	V.35 on data/clocks, V.28 on all other signals.
5 (0x05)	LM_449	RS449 (Identical to LM_530 setting).
6 (0x06)	LM_V28	RS232 (V.28).
7 (0x07)	LM_SHUTDOWN	All Inputs/Outputs on SP508 are Tri-stated.

Some of the above values can be combined with the following to enable Line Receiver and/or Line Driver "terminations" circuits inside the SP508 device. See the SP508 data sheet for details.

Value	Name	Description
8 (0x08)	LM_TERM	Termination enabled. <ul style="list-style-type: none"> <li>Line Receiver termination on V.35 and V.11 circuits</li> <li>Line Driver termination on V.35 circuits.</li> </ul>

### ***duplex\_mode***

This parameter sets up either Full, 2-wire half, or 4-wire half duplex operation of the electrical interface for the signals that connect to the 25D I/O connector. Only the Receiver Data and Transmitter Data signals are affected by this setting.

Value	Name	Description
0 (0x00)	DUPLEX_FULL	Both the receiver and the transmitter are always enabled.
1 (0x01)	TWO_WIRE_RTS	These modes perform the following action to the transceivers: <ul style="list-style-type: none"> <li>During transmission the transmitter is enabled and the receiver is disabled such that no characters are received during transmission.</li> <li>When not transmitting data the transmitter is disabled (tri-stated) and the receiver is enabled.</li> </ul>
17 (0x11)	TWO_WIRE_TXCOMP	
2 (0x02)	FOUR_WIRE_RTS	These modes perform the following action to the transceivers: <ul style="list-style-type: none"> <li>The receiver is always enabled.</li> <li>The transmitter is enabled during transmission and is otherwise disabled (tri-stated).</li> </ul>
18 (0x12)	FOUR_WIRE_TXCOMP	

The 2-wire and 4-wire settings differ only in the choice of signal which is used to control the duplexing operation, either RTS or TX Complete. RTS would be used if



software wanted to control the duplex operation. The TX complete is an IUSC device signal that can perform the duplex control in a more autonomous fashion.

### ***sync\_addr***

This is a collection of settings related to the operation of various "Character-oriented Synchronous" modes of operation.

#### ***tx0***

Sets the Transmitter Sync-0 character (for MonoSync and BiSync modes).

#### ***tx1***

Sets the Transmitter Sync-1 character (BiSync mode).

#### ***tx\_len***

Number of bits in Transmitter sync characters. ***tx\_len*** can be one of the following values:

<b>Value</b>	<b>Description</b>
0 (0x00)	Same number of bits as the TX data character length.
8 (0x08)	8 bits.

#### ***rx0***

Sets the Receiver Sync-0 character (for MonoSync and BiSync modes), or the first byte of an HDLC address in HDLC mode.

#### ***rx1***

Sets the Receiver Sync-1 character (BiSync mode), or the second byte of an HDLC address in HDLC mode.

#### ***rx\_len***

Number of bits in Receiver sync characters. ***rx\_len*** can be one of the following values:

<b>Value</b>	<b>Description</b>
0 (0x00)	Same number of bits as the RX data character length.
8 (0x08)	8 bits.

#### ***strip\_sync***

Any non-zero value will enable the stripping of SYNC characters from the receiver data stream. In MonoSync mode, a character that matches the value of rx0 is not placed in the receiver FIFO or included in any CRC calculations. In BiSync mode, a pair of consecutive characters that match rx0 and rx1 are not placed in the receiver FIFO or included in any CRC calculations.

#### ***dbits.tx/rx***

Sets the number of bits in each Data character.

#### ***tx\_frac\_stop***

Sets the number of Stop bits (in Async mode only), in units of 1/16 of a bit.

***tx\_frac\_stop*** can be from 9 to 32, inclusive.

#### ***parity.tx/rx***

Sets the use of parity bits on the Data characters. Parity bits can be enabled on Synchronous communications modes, but are usually used in Asynchronous communications. ***parity.tx/rx*** can be one of the following values:



Value	Name	Description
0 (0x00)	BHN_PAR_NONE	No parity.
1 (0x01)	BHN_PAR_EVEN	Even parity.
2 (0x02)	BHN_PAR_ODD	Odd parity.
3 (0x03)	BHN_PAR_SPACE	Space parity (0's).
4 (0x04)	BHN_PAR_MARK	Mark parity (1's).

### **crc**

This is a collection of settings related to the generation or checking of CRC values, that are commonly used with Synchronous protocols to validate data frames (or packets).

#### **tx\_type/rx\_type**

Sets the type of CRC to be employed. **tx\_type/rx\_type** can be one of the following values:

Value	Name	Description
0 (0x00)	CRC_NONE	Disable CRC.
1 (0x01)	CCITT	CRC-CCITT.
2 (0x02)	CRC16	CRC-16.
3 (0x03)	CRC32	CRC-32.

#### **tx\_start/rx\_start**

Sets the starting value for CRC computations. **tx\_start/rx\_start** can be one of the following values:

Value	Name	Description
0 (0x00)	CRC_START_0	Start with all zeros CRC.
1 (0x01)	CRC_START_1	Start with all ones CRC.

#### **tx\_mode**

Sets the conditions which cause the Transmitter to send a CRC. **tx\_mode** can be one or more of the following values:

Value	Name	Description
1 (0x01)	CRC_TX_UNDERRUN	CRC sent when Transmitter underruns.
2 (0x02)	CRC_TX_EOFM	CRC sent when EOF/EOM (end of frame, end of message) occurs.

#### **tx\_trig\_gate**

Sets the triggered or gated transmit mode. **tx\_trig\_gate** can be one of the following values:

Value	Name	Description
0 (0x00)	TX_TRIG_DISABLE	Triggered and gated transmit mode is disabled.
1 (0x01)	TX_TRIG_FALL	Hardware initiated by the falling edge of the 'tx_trig' signal.
2 (0x02)	TX_TRIG_RISE	Hardware initiated by the rising edge of the 'tx_trig' signal.
3 (0x03)	TX_GATED	Software initiated.

Settings **TX\_TRIG\_FALL** (falling edge) and **TX\_TRIG\_RISE** (rising edge) enable a mode where the Transmitter will only transmit 1 byte (character) each time the selected edge of a "triggering" signal occurs. This signal originates on the 25D I/O connector.

Setting **TX\_GATED** establishes a mode that allows software to quickly enable/disable the Transmitter via a signal that originates within the Main PLD logic. See **SET\_RTC\_GATE** and **SET\_RTC\_ALARM** messages for more information.

**rx\_gate**

Sets the gated receive mode. `rx_gate` can be one of the following values:

Value	Name	Description
0 (0x00)	RX_GATED_DISABLE	Gated receive mode is disabled.
1 (0x01)	RX_GATED	Software initiated.

Setting **RX\_GATED** establishes a mode that allows software to quickly enable/disable the Receiver via a signal that originates within the Main PLD logic. See **SET\_RTC\_GATE** and **SET\_RTC\_ALARM** messages for more information.

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_LINE2</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
9 (0x09)	BHN_ERROR_INVALID_PARM	There is an error or conflict in the specified port settings. See the <b>err_code</b> member of the response data for detailed information.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>SET_LINE2</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SET_LINE2</b> message.

*Data Length*

4 bytes

*Data*

```
struct bhn_lset2_stat
  int32  err_code
```

**err\_code**

Error code describing a port settings error or conflict.

**Requirements**

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[GET\\_LINE2](#)

**SET\_RTC\_ALARM**

The **SET\_RTC\_ALARM** message is used to configure alarms based on the RTC. These alarms start and stop the transmitter and/or the receiver.

**Message**

*Message Number*

55 (0x37)

**Port Number**

Number of port in which to set RTC alarms.

**Data Length**

6 bytes

**Data**

```
struct bhn_rtc_clock
  struct bhn_rtc_time
  | uint8  hours
  | uint8  minutes
  | uint8  seconds
  start_time

  struct bhn_rtc_time
  | uint8  hours
  | uint8  minutes
  | uint8  seconds
  stop_time
```

**start\_time/stop\_time**

When the RTC reaches start\_time, the transmitter and/or receiver is started. When the RTC reaches stop\_time, the transmitter and/or receiver is stopped.

In order for the alarms to operate on the transmitter the **SET\_LINE** message must first be sent with the **tx\_trig\_gate** member set to **TX\_GATED**. Similar to the transmitter, the alarms will operate on the receiver once the **SET\_LINE** message is sent with the **rx\_gate** member set to **RX\_GATED**.

**hours**

**hours** can be from 0 to 23, inclusive.

**minutes**

**minutes** can be from 0 to 59, inclusive.

**seconds**

**seconds** can be from 0 to 59, inclusive.

**Response****Status**

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_RTC_ALARM</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>SET_RTC_ALARM</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SET_RTC_ALARM</b> message.

**Data Length**

0 bytes

**Remarks**

To manually start and stop the transmitter and/or receiver, use the **SET\_RTC\_GATE** message.

**Requirements**

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[GET\\_RTC\\_ALARM SET\\_RTC\\_GATE SET\\_LINE](#)

**SET\_RTC\_CLOCK**

The **SET\_RTC\_CLOCK** message is used to change the time of the RTC.

**Message**

*Message Number*

53 (0x35)

*Port Number*

Number of port in which to change the RTC time.

*Data Length*

3 bytes

*Data*

```
struct bhn_rtc_clock
{
    struct bhn_rtc_time
    | uint8  hours
    | uint8  minutes
    | uint8  seconds
    time
}
```

**time**

**hours**

**hours** can be from 0 to 2,3 inclusive.

**minutes**

**minutes** can be from 0 to 59, inclusive.

**seconds**

**seconds** can be from 0 to 59, inclusive.

**Response**

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_RTC_CLOCK</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>SET_RTC_CLOCK</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SET_RTC_CLOCK</b> message.

*Data Length*

0 bytes



## Requirements

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

## See Also

[GET\\_RTC\\_CLOCK](#), [SET\\_RTC\\_ALARM](#), [SET\\_RTC\\_MODE](#)

## SET\_RTC\_GATE

The **SET\_RTC\_GATE** message is used to manually start or stop the transmitter and/or the receiver.

### Message

*Message Number*

57 (0x39)

*Port Number*

Number of port in which to start or stop.

*Data Length*

1 byte

*Data*

```
struct bhn_rtc_gate
uint8 state
```

#### **state**

If state is 0 then the transmitter and/or receiver is stopped. Otherwise, the transmitter and/or receiver is started.

In order for **SET\_RTC\_GATE** to operate on the transmitter the **SET\_LINE2** message must first be sent with the **tx\_trig\_gate** member set to **TX\_GATED**. Similar to the transmitter, the receiver gating will only operate once the **SET\_LINE2** message is sent with the **rx\_gate** member set to **RX\_GATED**.

### Response

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_RTC_GATE</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>SET_RTC_GATE</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SET_RTC_GATE</b> message.

*Data Length*

0 bytes

### Remarks

To automatically start and stop the transmitter and/or receiver based on the RTC time, use the **SET\_RTC\_ALARM**, **SET\_RTC\_CLOCK**, and **SET\_RTC\_MODE** messages.

**Requirements**

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**See Also**

[GET\\_RTC\\_GATE SET\\_RTC\\_ALARM SET\\_RTC\\_CLOCK SET\\_RTC\\_MODE SET\\_LINE2](#)

**SET\_RTC\_MODE**

The **SET\_RTC\_MODE** message is used to change the operational mode of the RTC.

**Message**

*Message Number*

60 (0x3C)

*Port Number*

Number of port in which to change the RTC mode.

*Data Length*

1 byte

*Data*

```
struct bhn_rtc_mode
  uint8 mode
```

**mode**

**mode** can be one or more of the following values:

Value	Name	Description
1 (0x01)	RTC_MINUTE_RISING	Minutes pulse on rising edge (otherwise falling edge).
2 (0x02)	RTC_EXT_TIMEBASE	Select external timebase (otherwise internal timebase).
4 (0x04)	RTC_SEC_RISING	Seconds pulse on rising edge (otherwise falling edge).
16 (0x10)	RTC_MINUTE_ONCE	Arm the mode where; The minute pulse is recognized only once, and the clock incrementing is masked until the minute pulse occurs. See also <a href="#">CANCEL_RTC_MPO</a> .

**Response**

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_RTC_MODE</b> message succeeded.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>SET_RTC_MODE</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SET_RTC_MODE</b> message.

*Data Length*

0 bytes



## Requirements

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

## See Also

[GET\\_RTC\\_MODE](#) [CANCEL\\_RTC](#) [MPO\\_SET\\_RTC](#) [CLOCK\\_SET\\_RTC](#) [ALARM](#)

## SET\_RTS

The **SET\_RTS** message is used to set the state of the RTS signal.

### Message

*Message Number*

15 (0x0F)

*Port Number*

Number of port in which to set the state of RTS.

*Data Length*

1 byte

*Data*

```
struct bhn_rtsdtr
uint8 onoff
```

#### **onoff**

If **onoff** is non-zero, **SET\_RTS** sets the RTS signal to a logical 1 (on). If **onoff** is zero, **SET\_RTS** sets the RTS signal to a logical 0 (off).

### Response

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_RTS</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

0 bytes

## Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

## See Also

[SET\\_RTS\\_DTR](#) [SET\\_DTR](#)



## SET\_RTS\_DTR

The **SET\_RTS\_DTR** message is used to set the state of the RTS and DTR signals.

### Message

*Message Number*

38 (0x26)

*Port Number*

Number of port in which to set the state of RTS and DTR.

*Data Length*

2 bytes

*Data*

```

struct bhn_rtsanddtr
  uint8  rts_onoff
  uint8  dtr_onoff

```

#### **rts\_onoff**

If **rts\_onoff** is non-zero, **SET\_RTS\_DTR** sets the RTS signal to a logical 1 (on). If **rts\_onoff** is zero, **SET\_RTS\_DTR** sets the RTS signal to a logical 0 (off).

#### **dtr\_onoff**

If **dtr\_onoff** is non-zero, **SET\_RTS\_DTR** sets the DTR signal to a logical 1 (on). If **dtr\_onoff** is zero, **SET\_RTS\_DTR** sets the DTR signal to a logical 0 (off).

### Response

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SET_RTS_DTR</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

0 bytes

### Requirements

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

### See Also

[SET\\_DTR](#) [SET\\_RTS](#)

## START\_STOP\_BREAK

The **START\_STOP\_BREAK** message is used to start and stop transmission of a break condition.

### Message

*Message Number*

14 (0x0E)



*Port Number*

Number of port in which to start or stop transmitting a break condition.

*Data Length*

1 byte

*Data*

```
struct bhn_ssbreak
  uint8  onoff
```

**onoff**

If **onoff** is non-zero, **START\_STOP\_BREAK** starts transmitting a break condition. If **onoff** is zero, **START\_STOP\_BREAK** stops transmitting a break condition.

**Response**

*Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>START_STOP_BREAK</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.

*Data Length*

0 bytes

**Requirements**

<b>Protocol version</b>	9 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	No
<b>Port state must be</b>	Open

**See Also**

[SEND\\_BREAK](#)

## SYNC\_HUNT

The **SYNC\_HUNT** message is used to force the receiver into "Hunt Mode" immediately, regardless of its previous state.

**Message**

*Message Number*

52 (0x34)

*Port Number*

Number of port in which to put the receiver into "Hunt Mode".

*Data Length*

0 bytes

**Response***Status*

Status Value	Status Name	Description
0 (0x00)	CMD_OK	The <b>SYNC_HUNT</b> message succeeded.
3 (0x03)	BHN_ERROR_UNSPECIFIED	A non categorized error occurred.
11 (0x0B)	BHN_ERROR_PROTO_LEN	The data length specified in the message header is invalid.
13 (0x0D)	BHN_ERROR_CLOSED	The port is currently closed, but <b>SYNC_HUNT</b> requires it to be open.
14 (0x0E)	BHN_ERROR_HW_TYPE	The underlying port hardware does not support the <b>SYNC_HUNT</b> message.

*Data Length*

0 bytes

**Remarks**

In Synchronous modes, "Hunt Mode" means that the receiver starts searching for a Sync or Flag sequence. In Asynchronous modes it starts searching for a start bit. In any mode, the receiver discards any partial character that was in progress when the **SYNC\_HUNT** message is issued.

**Requirements**

<b>Protocol version</b>	10 or greater
<b>General message</b>	No
<b>Port message</b>	Yes
<b>Async</b>	Yes
<b>Sync</b>	Yes
<b>Port state must be</b>	Open

**3.3 Director Summary**

Director Number	Director Name	Director Length <sup>1</sup>	Protocol Version <sup>2</sup>
1 (0x01)	<b>START_OF_FRAME</b>	2+	10+
2 (0x02)	<b>DATA</b>	1+	10+
3 (0x03)	<b>END_OF_FRAME</b>	0+	10+
5 (0x05)	<b>EVENT2</b>	25+	10+

**Table 3-2**

<sup>1</sup> Director length is in bytes.

<sup>2</sup> Protocol version refers to the earliest version that the director is supported. Versions earlier than 9 are no longer used and this document does not cover them.



## 3.4 Director Detail

### DATA

The **DATA** director is used to transport intermediate frame data. Intermediate frame data occurs between **START\_OF\_FRAME** and **END\_OF\_FRAME** directors.

#### Director

*Director Number*

2 (0x02)

*Data Length*

1+ bytes (max 65535 bytes)

*Data*

The data area of the **DATA** director consists of purely serial data.

#### Remarks

A **DATA** director with a length of zero is not legal.

#### Requirements

Protocol version	10 or greater
------------------	---------------

#### See Also

[START\\_OF\\_FRAME](#) [END\\_OF\\_FRAME](#)

### EVENT2

The **EVENT2** director is used to report the occurrence of events.

#### Director

*Director Number*

5 (0x05)

*Data Length*

25+ bytes (max 65535 bytes)

*Data*

The data area of the **EVENT2** director consists of a header and optional serial data. If the length of the **EVENT2** director is greater than the length of the header (greater than 25 bytes), serial data follows the header (max serial data length is 65535-25=65510 bytes). Otherwise, there is no serial data associated with this director. The structure **bhn\_dir\_event2** represents the header portion of the **EVENT2** director.

```
struct bhn_dir_event2
{
    uint16 id
    struct bhn_event2_occurred
    | uint16 events
    | uint32 events2
    | uint8 msr
    | int32 oe_count
    | int32 pe_count
    | int32 fe_count
    | int32 brk_count
    evso
}
```

***id***

If the events are associated with a transmitting frame, ***id*** is the application specific value specified in the ***id*** member of the frame's **START\_OF\_FRAME** director.

***evso***

Same as data structure of **EVENT2\_OCCURRED** message.

**Requirements**

Protocol version	10 or greater
------------------	---------------

**See Also**

[EVENT2\\_OCCURRED START\\_OF\\_FRAME](#)

---

**END\_OF\_FRAME**

The **END\_OF\_FRAME** director is used to mark the ending of a frame of data.

**Director**

*Director Number*

3 (0x03)

*Data Length*

0+ bytes (max 65535 bytes)

*Data*

The data area of the **END\_OF\_FRAME** director consists of optional serial data. If the length of the **END\_OF\_FRAME** director is greater than zero, serial data follows. Otherwise, there is no serial data associated with this director.

**Requirements**

Protocol version	10 or greater
------------------	---------------

**See Also**

[START\\_OF\\_FRAME DATA](#)

---

**START\_OF\_FRAME**

The **START\_OF\_FRAME** director is used to mark the beginning of a frame of data.

**Director**

*Director Number*

1 (0x01)

*Data Length*

2+ bytes (max 65535 bytes)

*Data*

The data area of the **START\_OF\_FRAME** director consists of a header and optional serial data. If the length of the **START\_OF\_FRAME** director is greater than the length of the header (greater than 2 bytes), serial data follows the header (max serial data length is 65535-2=65533 bytes). Otherwise, there is no serial data associated with this director. The structure **bhn\_dir\_sof** represents the header portion of the **START\_OF\_FRAME** director.

```
struct bhn_dir_sof
  uint16 id
```

***id***

When the **START\_OF\_FRAME** director is used in the receive side of a data channel, the ***id*** member has no meaning. However, when used in the transmit side of a data channel, the ***id*** member can contain any host application specific 16 bit value.

**Requirements**

Protocol <b>version</b>	10 or greater
-------------------------	---------------

**See Also**

[DATA END OF FRAME](#)



## 4 Index

### C

CANCEL\_RTC\_MPO, 13  
channel, 4  
CLOSE\_PORT, 8, 14

### D

DATA, 8, 70  
data format, 6  
data mode, 6  
    packet, 7, 8  
    stream, 6  
    stream, 8  
director, 4, 7  
    DATA, 70  
    director data, 7  
    director number, 7  
    END\_OF\_FRAME, 71  
    EVENT2, 70  
    length, 7  
    START\_OF\_FRAME, 71

### E

END\_OF\_FRAME, 8, 71  
EVENT, 8  
EVENT\_OCCURRED, 14  
EVENT2, 70  
EVENT2\_OCCURRED, 16

### G

GET\_BHN\_INFO, 17  
GET\_DATA\_MODE, 18  
GET\_EVENT, 19  
GET\_EVENT2, 20  
GET\_FIFO\_CONTROL, 21  
GET\_LINE, 22  
GET\_LINE\_STATE, 24  
GET\_LINE2, 23  
GET\_MSR, 25  
GET\_PORTCAP, 26  
GET\_PROTO\_SUPPORT, 9, 27  
GET\_PROTO\_VER, 28  
GET\_PSON, 29  
GET\_RTC\_ALARM, 30  
GET\_RTC\_CLOCK, 31  
GET\_RTC\_GATE, 32  
GET\_RTC\_MODE, 32

### M

message, 4  
    CANCEL\_RTC\_MPO, 13

CLOSE\_PORT, 14  
    data, 5  
EVENT\_OCCURRED, 14  
EVENT2\_OCCURRED, 16  
GET\_BHN\_INFO, 17  
GET\_DATA\_MODE, 18  
GET\_EVENT, 19  
GET\_EVENT2, 20  
GET\_FIFO\_CONTROL, 21  
GET\_LINE, 22  
GET\_LINE\_STATE, 24  
GET\_LINE2, 23  
GET\_MSR, 25  
GET\_PORTCAP, 26  
GET\_PROTO\_SUPPORT, 27  
GET\_PROTO\_VER, 28  
GET\_PSON, 29  
GET\_RTC\_ALARM, 30  
GET\_RTC\_CLOCK, 31  
GET\_RTC\_GATE, 32  
GET\_RTC\_MODE, 32  
    length, 5  
    message number, 5  
OPEN\_PORT, 33  
PING, 34  
    port number, 5  
PURGE, 35  
RESET\_BHN, 36  
RESUME\_TX, 36  
SEND\_BREAK, 37  
SEND\_IMMEDIATE, 38  
SEND\_XCHAR, 38  
SET\_DATA\_MODE, 39  
SET\_DTR, 40  
SET\_EVENT, 41  
SET\_EVENT2, 42  
SET\_FIFO\_CONTROL, 45  
SET\_LINE, 47  
SET\_LINE2, 49  
SET\_RTC\_ALARM, 61  
SET\_RTC\_CLOCK, 63  
SET\_RTC\_GATE, 64  
SET\_RTC\_MODE, 65  
SET\_RTS, 66  
SET\_RTS\_DTR, 67  
START\_STOP\_BREAK, 67  
SYNC\_HUNT, 68  
message format, 5

### O

OPEN\_PORT, 7, 8, 33

### P

packet mode, 4



PING, 34  
 protocol usage, 7  
 PURGE, 35

## R

RESET\_BHN, 36  
 response, 4  
   CANCEL\_RTC\_MPO, 13  
   CLOSE\_PORT, 14  
   completed message number, 6  
   data, 6  
   EVENT\_OCCURRED, 15  
   EVENT2\_OCCURRED, 16  
   GET\_BHN\_INFO, 17  
   GET\_DATA\_MODE, 19  
   GET\_EVENT, 20  
   GET\_EVENT2, 21  
   GET\_FIFO\_CONTROL, 22  
   GET\_LINE, 23  
   GET\_LINE\_STATE, 24  
   GET\_LINE2, 24  
   GET\_MSR, 25  
   GET\_PORTCAP, 26  
   GET\_PROTO\_SUPPORT, 27  
   GET\_PROTO\_VER, 28  
   GET\_PSON, 29  
   GET\_RTC\_ALARM, 30  
   GET\_RTC\_CLOCK, 31  
   GET\_RTC\_GATE, 32  
   GET\_RTC\_MODE, 33  
   length, 6  
   OPEN\_PORT, 34  
   PING, 35  
   port number, 6  
   PURGE, 35  
   RESET\_BHN, 36  
   response number, 6  
   RESUME\_TX, 37  
   SEND\_BREAK, 37  
   SEND\_IMMEDIATE, 38  
   SEND\_XCHAR, 39  
   SET\_DATA\_MODE, 40  
   SET\_DTR, 41  
   SET\_EVENT, 42  
   SET\_EVENT2, 44  
   SET\_FIFO\_CONTROL, 46  
   SET\_LINE, 48  
   SET\_LINE2, 61  
   SET\_RTC\_ALARM, 62  
   SET\_RTC\_CLOCK, 63  
   SET\_RTC\_GATE, 64  
   SET\_RTC\_MODE, 65  
   SET\_RTS, 66  
   SET\_RTS\_DTR, 67  
   START\_STOP\_BREAK, 68  
   status, 6  
   SYNC\_HUNT, 69  
 response format, 6

RESUME\_TX, 36

## S

SEND\_BREAK, 37  
 SEND\_IMMEDIATE, 38  
 SEND\_XCHAR, 38  
 SET\_DATA\_MODE, 39  
 SET\_DTR, 40  
 SET\_EVENT, 41  
 SET\_EVENT2, 42  
 SET\_FIFO\_CONTROL, 45  
 SET\_LINE, 47  
 SET\_LINE2, 49  
 SET\_RTC\_ALARM, 61  
 SET\_RTC\_CLOCK, 63  
 SET\_RTC\_GATE, 64  
 SET\_RTC\_MODE, 65  
 SET\_RTS, 66  
 SET\_RTS\_DTR, 67  
 START\_OF\_FRAME, 8, 71  
 START\_STOP\_BREAK, 67  
 stream mode, 4  
 structures  
   bhn\_break, 37  
   bhn\_data\_mode, 19, 39  
   bhn\_dir\_event2, 70  
   bhn\_dir\_sof, 71  
   bhn\_event, 20, 41  
   bhn\_event\_occurred, 15  
   bhn\_event2, 21, 43  
   bhn\_event2\_occurred, 16  
   bhn\_fifo, 22, 45  
   bhn\_info, 17  
   bhn\_info\_cmd, 17  
   bhn\_lset, 47  
   bhn\_lset2, 49  
   bhn\_lset2\_stat, 61  
   bhn\_lstate, 25  
   bhn\_msr, 25, 26  
   bhn\_open, 34  
   bhn\_proto\_supp, 27  
   bhn\_proto\_ver, 28  
   bhn\_pson, 29  
   bhn\_purge, 35  
   bhn\_rtc\_clock, 62, 63  
   bhn\_rtc\_gate, 64  
   bhn\_rtc\_mode, 33, 65  
   bhn\_rtsanddtr, 67  
   bhn\_rtsdtr, 40, 66  
   bhn\_sendi, 38  
   bhn\_ssbreak, 68  
   bhn\_xchar, 38  
 SYNC\_HUNT, 68

## T

TCP  
 connections, 4



ports, 9