LORD MANUAL

# Wireless Sensor Networks

**LXRS® Data Communications Protocol**


LORD MicroStrain®
SENSING SYSTEMS

LORD MicroStrain®
SENSING SYSTEMS



MicroStrain® Sensing Systems
459 Hurricane Lane
Suite 102
Williston, VT 05495
United States of America
Phone: 802-862-6629
Fax: 802-863-4093

www.microstrain.com
support@microstrain.com

**REVISED: July 9, 2013**

**Information subject to change**
Information in this document is subject to change without notice and does not represent a commitment on the part of LORD. While LORD makes every effort as to the accurateness of this document, it assumes no responsibility for errors or omissions.

**SDK**
The Software Development Kit (SDK) is provided “as is” and any expressed or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall LORD or its contributors be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data, or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this SDK, even if advised of the possibility of such damage. LORD will make every effort to amplify the instructions contained in the Data Communications Protocol manual and sample code but will neither undertake to detail the functioning of the hardware or firmware in the Wireless Sensor Networks family nor debug the purchaser’s code.

# Table of Contents

# Overview

This document describes the data communications protocol for the LORD Wireless Sensor Networks LXRS™ product line which includes the following devices:

| Device | Firmware Version |
|---|---|
| V-Link® -LXRS™ Wireless Voltage Node | 8.00+ |
| G-Link® -LXRS™ Wireless Accelerometer Node | 8.00+ |
| SG-Link® -LXRS™ Wireless Strain Node | 8.00+ |
| DVRT-Link™ (LXRS™) Wireless Displacement Node | 8.00+ |
| ENV-Link™ -Mini -LXRS™ Wireless Environmental Monitoring Node | 8.02+ |
| WSDA® -Base-101 Analog Output Base Station | 3.00+ |
| WSDA® -Base-102 RS-232 Serial Output Base Station | 3.00+ |
| WSDA® -Base-104 USB Base Station | 3.00+ |
| WSDA® -1000 Gateway | 2.1.12+ |

If your equipment has firmware earlier than stated in the above table, there may be a requirement to upgrade in order to take full advantage of this protocol.

**USB Interface**

When communicating with a WSDA-Base via USB, a virtual serial port is created using the Silicon Laboratories Virtual Communications Port (VCP) driver. Please see the section entitled CP210x USB to UART Bridge Controller for more details.

| USB Virtual Communication Port Configuration | |
|---|---|
| Baud Rate | 921, 600 |
| Parity | None |
| Data Bits | 8 |
| Stop Bits | 1 |

**RS-232 Interface**

When communication between the WSDA-Base and the host computer is via an RS-232 connection, the physical layer provides the interface and there is no driver required by the base station.

| RS-232 Port Configuration | |
|---|---|
| Baud Rate | 115,200 (default) or 921,600 |
| Parity | None |
| Data Bits | 8 |
| Stop Bits | 1 |

# Commands Overview

The following is a quick reference list of commands for controlling the Base Station and the wireless sensor Nodes.

Each command has its bytes listed in the form: *[# of bytes] Value*
Please see the sections following for more detailed information on each command.

## Base Station Commands Reference

**Ping Base**
[1] 0x01

**Read Base EEPROM**
[1] 0x73
[2] EEPROM Address
[2] Checksum

**Write Base EEPROM**
[1] 0x78
[2] EEPROM Address
[2] Value to Write
[2] Checksum

**Enable Beacon**
[2] 0xBEAC
[4] Timestamp

**Disable Beacon**
[2] 0xBEAC
[4] 0xFFFFFFFF

## Node Commands Reference

### Short Ping
[1] 0x02
[2] Node Address

### Long Ping
[1] 0xAA
[2] 0x0500
[2] Node Address
[3] 0x020002
[2] Checksum

### Read Node EEPROM
[1] 0xAA
[2] 0x0500
[2] Node Address
[3] 0x040003
[2] EEPROM Address
[2] Checksum

### Write Node EEPROM
[1] 0xAA
[2] 0x0500
[2] Node Address
[3] 0x060004
[2] EEPROM Address
[2] Value to Write
[2] Checksum

### Sleep
[1] 0x32
[2] Node Address

### Stop Node
[1] 0xAA
[2] 0xFE00
[2] Node Address
[3] 0x020090
[2] Checksum

### Arm for Datalogging
[1] 0xAA
[2] 0x0500
[2] Node Address
[1] 0x02 + Text Len
[1] 0x00
[1] 0x0D
[0-50] User Bytes
[2] Checksum

### Trigger Datalogging
[1] 0xAA
[2] 0x0500
[2] Node Address
[3] 0x0A000E
[4] Timestamp (sec)
[4] Timestamp (nano)
[2] Checksum

### Download Page
[1] 0x05
[2] Node Address
[2] Page Index

### Erase Data
[1] 0x06
[2] Node Address
[4] 0x08100CFF

### Streaming
[1] 0x38
[2] Node Address

### Low Duty Cycle
[1] 0xAA
[2] 0x0500
[2] Node Address
[3] 0x020038
[2] Checksum

### Sync Sampling
[1] 0xAA
[2] 0x0500
[2] Node Address
[3] 0x02003B
[2] Checksum

### Read Single Sensor
[1] 0x03
[2] Node Address
[1] 0x01
[1] Channel #

### Auto Balance
[1] 0x62
[2] Node Address
[1] Channel #
[2] Target Balance

## Data Format

Throughout this document, the following values will be used to represent their corresponding data types:

| Value | Data Type |
|---|---|
| uint8 | 1 byte unsigned integer |
| uint16 | 2 byte unsigned integer |
| uint32 | 4 byte unsigned integer |
| int8 | 1 byte signed integer |
| float | 4 byte float |

All commands, responses, and data are in **big endian**. Therefore, when building a field that is more than 1 byte, the Most Significant Byte (MSB) is the first byte, and the Least Significant Byte (LSB) is the last byte.

### 2-Byte Value Example [uint16]

A value of 476 would yield an MSB of 1 and an LSB of 220.

| | Decimal | Hex | Binary |
|---|---|---|---|
| **2 Byte Value** | 476 | 01 DC | 00000001 1101100 |
| **MSB** | 1 | 01 | 00000001 |
| **LSB** | 220 | DC | 1101100 |

### Sample C++ Code

```
typedef unsigned char BYTE;
typedef unsigned short WORD;

void ValueToMSBandLSB(WORD value, BYTE& msb, BYTE& lsb)
{
    msb = value >> 8;     //Shift 8 bits right, drop the lower byte.
    lsb = value & 0x00ff; //Mask out the upper byte.
}

WORD ValueFromMSBandLSB(BYTE msb, BYTE lsb)
{
    return (msb << 8)| lsb; //Shift msb 8 bits left, OR the lsb in.
}
```

### 4-Byte Value Example [uint32]

A value of 1,326,214,446 would yield the following:

| | Decimal | Hex | Binary |
|---|---|---|---|
| **4 Byte Value** | 1,326,214,446 | 4F 0C 6D 2E | 01001111 00001100 01101101 00101110 |
| **Byte 1 (MSB)** | 79 | 4F | 01001111 |
| **Byte 2** | 12 | 0C | 00001100 |
| **Byte 3** | 109 | 6D | 01101101 |
| **Byte 4 (LSB)** | 46 | 2E | 00101110 |

### Sample C++ Code

```
typedef unsigned short WORD;
typedef unsigned long DWORD;

void ValueTo4Bytes(DWORD value, BYTE& b1, BYTE& b2, BYTE& b3, BYTE& b4)
{
    b1 = value >> 24;          //Shift 24 bits right, drop lower bytes
    b2 = (value >> 16) & 0xff; //Shift 16 bits right, drop lower bytes, Mask out the upper bytes

    b3 = (value >> 8) & 0xff;  //Shift 8 bits right, drop lower byte, Mask out the upper bytes
    b4 = value & 0xff;         //Mask out the upper bytes
}

DWORD ValueFrom4Bytes(BYTE b1, BYTE b2, BYTE b3, BYTE b4)
{
    WORD hiWord = (b1 << 8)| b2;    //Shift msb 8 bits left, OR the lsb in.
    WORD loWord = (b3 << 8)| b4;    //Shift msb 8 bits left, OR the lsb in.

    return (hiWord << 16) | loWord; //Shift hiWord 16 bits left, OR the loWord in.
}
```

Many programming languages can directly interpret a single byte as a *signed* 8-bit value. For example:

```
typedef unsigned char BYTE;

int InterpretAsSigned(BYTE value)
{
    return static_cast<signed char>value;
}
```

When it is not possible to interpret the byte as a signed 8-bit value, the following function will perform the 2's complement conversion from unsigned to signed:

```
typedef unsigned char BYTE;

int InterpretAsSigned(BYTE value)
{
    int result;

    //if the byte value is < 128
    if(value < 128)
    {
        result = value;
    }
    //if the byte value is >= 128
    else
    {
        //take the (value mod 128) - 128
        result = (value % 128) - 128;
    }

    //return the result as an integer
    return result;
}
```

## RSSI

The Received Signal Strength Indicator (RSSI) is a measurement of the power present in a received radio signal. Node commands such as Long Ping and Node Discovery return a true RSSI value, measured in dBm, in their response packet. This signed 1-byte value can range from -95 dBm to +5 dBm.

Some command responses return two RSSI values, **Node RSSI** and **Base Station RSSI**. The **Node RSSI** is the signal strength that the node received the command from the base station. The **Base Station RSSI** is the signal strength that the base station received the response back from the node.

## Checksum

Some commands and responses require or supply a checksum. The checksum is used to ensure that the data was transmitted without error. The documentation for each command or response that uses a checksum will detail how the checksum is calculated. The checksum is calculated by summing all the bytes protected by the checksum and taking the modulo N of the sum, where N is 256 for a 1 byte checksum and 65536 for a 2 byte checksum.

$$N\ Byte\ Checksum = \left(\sum_{i=x}^{I} Byte[i]\right)\ mod\ (256^N), \qquad N \in [1,2]$$

**Single Byte Checksum Example:**

Protected Command Bytes

| Byte | Decimal Value |
|---|---|
| Byte 1 | 10 |
| Byte 2 | 121 |
| Byte 3 | 37 |
| Byte 4 | 235 |

Sum: `403 = 10 + 121 + 37 + 235`
Checksum: `147 = 403 MOD 256`

**Two Byte Checksum Example:**

Protected Command Bytes

| Byte | Decimal Value |
|---|---|
| Byte 1 | 10 |
| Byte 2 | 121 |
| Byte 3 | 37 |
| Byte 4 | 235 |

Sum: `403 = 10 + 121 + 37 + 235`
Checksum: `403 = 403 MOD 65536`

Splitting the checksum into 2 bytes is done by the following:

Checksum MSB: `1 = 403 / 256`
Checksum LSB: `147 = 403 MOD 256`

## Packet Structure

Most command, response, and data packets found in this document follow a certain packet structure:

| Value | # of Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| [0x0 - 0xFF] | 1 | uint8 | Delivery Stop Flag |
| [0x0 - 0xFF] | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| [0x0 - 0x6A] | 1 | uint8 | Payload Length |
| | 2 - 106 | | Payload |
| [0x0 - 0xFF] | 1 | int8 | Node RSSI |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Start of Packet:** indicates the start of a new packet, and is always 0xAA.
**Delivery Stop Flag:** indicates where the packet was intended to go.
**App Data Type:** indicates the type of the packet.
**Node Address:** the address of the Node that the packet is from or destined to.
**Payload Length:** the number of bytes in the payload (the bytes immediately following this byte).
**Payload:** contains up to 106 bytes of information.

**Node RSSI:** indicates the Node's RSSI.
**Base RSSI:** indicates the Base Station's RSSI.
**Checksum:** the checksum of the packet (from the Delivery Stop Flag byte to the last Payload byte).

Possible App Data Types:

| App Data Type | Description |
|---|---|
| 0x04 | Low Duty Cycle Data Packet |
| 0x0A | Synchronized Sampling Data Packet |
| 0x11 | Diagnostic Packet |
| 0x0E | Asynchronous Digital Only Data Packet |
| 0x0F | Asynchronous Digital & Analog Data Packet |

*See also:* RSSI, Checksum

# Base Station Commands

## Ping Base Station

The **Ping Base Station** command is used to ensure that the host computer and the Base Station are properly communicating.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x01 | 1 | uint8 | Command ID |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x01 | 1 | uint8 | Command ID |

**Fail Response:** No Response

## Read Base Station EEPROM

The **Read Base Station EEPROM** command is used to read the value of a specific memory address from the Base Station's EEPROM. See the *Base Station EEPROM Map* for specific memory address details.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x73 | 1 | uint8 | Command ID |
| [0x0 - 0x400] | 2 | uint16 | EEPROM address to read |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x73 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Value Read from Base Station EEPROM |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Fail Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x21 | 1 | uint8 | Fail Response ID |

*See also:* Data Format, Checksum

## Write Base Station EEPROM

The **Write Base Station EEPROM** command is used to write a value to a specific memory address on the Base Station's EEPROM. The **Read Base Station EEPROM** command may be used to further verify that the value was written correctly.
See *Base Station EEPROM Map* for specific memory address details.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x78 | 1 | uint8 | Command ID |
| [0x0 - 0x400] | 2 | uint16 | EEPROM address to write to |
| [0x0 - 0xFFFF] | 2 | uint16 | Value to write to the EEPROM |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x78 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Value Written to the Base Station's EEPROM |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Fail Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x21 | 1 | uint8 | Fail Response ID |

*See also*: Data Format, Checksum

## Enable Beacon

The **Enable Beacon** command is used to turn on the beacon on the Base Station. The beacon is used to synchronize and start a group of nodes when performing **Synchronized Sampling**.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xBEAC | 2 | uint16 | Command ID |
| [0x0 - 0xFFFFFFFE] | 4 | uint32 | Timestamp to give to the beacon |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xBEAC | 2 | uint16 | Command ID |

**Fail Response:** No Response

**Command Packet Timestamp Bytes**
The last 4 bytes in the Command Packet are Timestamp bytes. The beacon that is started from this command will send a Timestamp beginning from this value. These bytes represent the current UTC time in seconds from the Unix Epoch (January 1, 1970).

*See also*: Synchronized Sampling

## Disable Beacon

The **Disable Beacon** command is used to turn off the beacon on the Base Station. Sending the **Stop Node** command from the Base Station that is beaconing will automatically turn off the beacon as well.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xBEAC | 2 | uint16 | Command ID |
| 0xFFFFFFFF | 4 | uint32 | Value to Disable the Beacon |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xBEAC | 2 | uint16 | Command ID |

**Fail Response:** No Response

Note: Notice that the **Disable Beacon** command packet is the same as the **Enable Beacon** command packet, but with 0xFFFFFFFF for its “Timestamp.”

*See also*: Synchronized Sampling, Stop Node

# Node Commands

## Short Ping

The **Short Ping** command is used to check the communication between the Base Station and the Node.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x02 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x02 | 1 | uint8 | Command ID |

**Fail Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x21 | 1 | uint8 | Fail Response ID |

Note: The **Short Ping** command is handled directly by the base station and will always give back an immediate response of whether the Node can be communicated with or not. Some other commands do not have a failure response, which requires the user wait for an amount of time to determine the command has failed. Therefore, the **Short Ping** command can be useful for checking the communication with a node before sending other commands.

*See also:* Data Format

## Long Ping

The **Long Ping** command is used to check the communication between the Base Station and the Node, and gives more information than the Short Ping command.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x05 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x02 | 1 | uint8 | Payload Length |
| 0x0002 | 2 | uint16 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Initial Response:**
An initial response comes directly from the Base Station to acknowledge that the command was received by the Base Station and sent to the Node.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Packet-Sent Acknowledgement |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x07 | 1 | uint8 | Delivery Stop Flag |
| 0x02 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x02 | 1 | uint8 | Payload Length |
| 0x0000 | 2 | uint16 | Empty Payload (not used) |
| [0x0 - 0xFF] | 1 | int8 | Node RSSI |
| [0x0 - 0xFF] | 1 | int8 | Base RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Fail Response:** No Response

*See also:* Data Format, Checksum

## Read Node EEPROM

The **Read Node EEPROM** command is used to read the value of a specific memory address from the Node's EEPROM. See the *Node EEPROM Map* for specific memory address details.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x05 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x04 | 1 | uint8 | Payload Length |
| 0x0003 | 2 | uint16 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | EEPROM Address to read |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Initial Response:**
An initial response comes directly from the Base Station to acknowledge that the command was received by the Base Station and sent to the Node.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Packet-Sent Acknowledgement |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x00 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x02 | 1 | uint8 | Payload Length |
| [0x0 - 0xFFFF] | 2 | uint16 | Value read from EEPROM |
| [0x0 - 0xFF] | 1 | int8 | RESERVED |
| [0x0 - 0xFF] | 1 | int8 | Base RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Fail Response:** No Response

*See also*: Data Format, Checksum

## Write Node EEPROM

The **Write Node EEPROM** command is used to write a value to a specific memory address on the Node's EEPROM. The Read Node EEPROM command can be used to further verify the value was written correctly. See the *Node EEPROM Map* for specific memory address details.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x05 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x06 | 1 | uint8 | Payload Length |
| 0x0004 | 2 | uint16 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | EEPROM Address to write to |
| [0x0 - 0xFFFF] | 2 | uint16 | Value to write to EEPROM |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |


**Initial Response:**
An initial response comes directly from the Base Station to acknowledge that the command was received by the Base Station and sent to the Node.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Packet-Sent Acknowledgement |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x00 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x02 | 1 | uint8 | Payload Length |
| 0x0004 | 2 | uint16 | Command ID |
| [0x0 - 0xFF] | 1 | int8 | RESERVED |
| [0x0 - 0xFF] | 1 | int8 | Base RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |


**Fail Response:** No Response

*See also*: Data Format, Checksum

## Initiate Sleep Mode

The **Initiate Sleep Mode** command is used to put the Node in a low power state. When the Node is in this low power Sleep Mode, it will not hear any commands except for the Stop Node command, which will wake the node and put it back into its normal, idle state. The Node should be put into Sleep Mode when the user no longer needs to communicate with the node, but wishes to keep it powered on and preserve battery life.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x32 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |

**Success Response:** No Response
**Fail Response:** No response

### Waking a Node

A Node in Sleep Mode periodically awakes, listens for a Stop Node command, and if none is received, returns to sleep. To wake a sleeping Node, send the Stop Node command.

### Sleep Interval

The Node's Sleep Interval is the interval at which the Node will awake and listen for the Stop Node command. This can be configured with EEPROM 66.

### User Inactivity Timeout

The Node's User Inactivity Timeout is the length of time, without user activity, before the Node enters Sleep Mode. After this length of time has expired, the Node will automatically enter Sleep Mode, and must be woken using the Stop Node command as usual before any other communication will occur. This can be configured with EEPROM 70.

### Relevant Memory Locations

Node EEPROM 66: Sleep Interval
Node EEPROM 70: User Inactivity Timeout

*See also:* Data Format, Short Ping, Stop Node

## Stop Node

The **Stop Node** command is used to stop a Node that is in any type of sampling mode, or wake a Node that is in Sleep Mode.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0xFE | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x02 | 1 | uint8 | Payload Length |
| 0x0090 | 2 | uint16 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Initial Response:**
An initial response comes directly from the Base Station to acknowledge that the command was received by the Base Station and sent to the Node.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Packet-Sent Acknowledgment |

**Success Response:**
The Node was stopped and will now communicate.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x90 | 1 | uint8 | Command ID |
| 0x01 | 1 | uint8 | Base Station Ceased Stop Node |

**Fail Response:**
The **Stop Node** process was aborted by user sending any byte to the Base Station.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x21 | 1 | uint8 | Fail/Abort Response ID |
| 0x01 | 1 | uint8 | Base Station Ceased Stop Node |

The **Stop Node** command puts the Base Station in a mode that sends small packets as fast as possible in an attempt to communicate with the Node. The Base Station will periodically check, after 1000 packets, to see if the Node has responded to a ping request. The function will continue indefinitely until either the Node responds, or the user sends any byte to the Base Station. The base station will send a 0x01 response when it has ceased sending the **Stop Node** packets, for both success and fail.

### Broadcast Special Case

When the broadcast Node address 65535 (0xFFFF) is used, the Base Station does not check for a ping response. It will continue sending the stop Node command until interrupted by the user (any single byte sent to the Base Station). This will attempt to stop **all** nodes on the current frequency that the Base Station is on.

*See also:* Data Format, Checksum

## Arm Node (for Datalogging)

Use the **Arm Node** command to put the node in an armed state waiting for the **Trigger Armed Datalogging** command. A Node will stay in this armed state for a default of 10 seconds (EEPROM adjustable). To disarm an Armed Node, use the **Stop Node** command.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x05 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x02 + # user bytes | 1 | uint8 | Payload Length |
| 0x000D | 2 | uint16 | Command ID |
| | 0 - 50 | any | User entered data (if any) |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Initial Response:**
An initial response comes directly from the Base Station to acknowledge that the command was received by the Base Station and sent to the Node.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Packet-Sent Acknowledgement |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x07 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x03 | 1 | uint8 | Payload Length |
| 0x000D | 2 | uint16 | Command ID |
| 0x00 | 1 | uint8 | RESERVED (Fail Type) |
| [0x0 - 0xFF] | 1 | int8 | RESERVED |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Fail Response:** No Response

### User Entered Data

The user may send up to 50 bytes with the Arm Command which will then be in the header information of the packet when the data is downloaded from the node using the **Download Page** command.

### Relevant Memory Locations

Node EEPROM 270: Armed Datalogging Timeout

*See also:* Data Format, Checksum, Trigger Armed Datalogging, Download Page

## Trigger Armed Datalogging

The **Trigger Armed Datalogging** command is used to initiate a data capture session on-board the Node. The data will be stored in the Node's 2MB memory and may be downloaded at a later time. The **Trigger Armed Datalogging** command can be sent to the broadcast node address of 65535. This will be sent to all nodes on the operating frequency, and any nodes that are in the "Armed" state will start datalogging. This can be useful to start datalogging on multiple nodes at the same time.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x05 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x0A | 1 | uint8 | Payload Length |
| 0x000E | 2 | uint16 | Command ID |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (seconds) |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (nanoseconds) |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Success Response**: No Response
**Fail Response:** No Response

### Timestamp

The Timestamp is made up of 2 values, each of which are 4 bytes. The first value represents the seconds of the timestamp, and the second value represents the nanoseconds of the timestamp. When downloading this data from the node, this timestamp will be transmitted in the header information and can be used to find the exact timestamp of each data point.
These bytes represent the current UTC time in seconds from the Unix Epoch (January 1, 1970).

### Notes

- If a Sensor Event Driven Trigger (SEDT) is being used, the equivalent of a trigger command is automatically issued within the Node's firmware as a result of an output ceiling, or output floor being reached. Note that if SEDT is enabled, the Node will *not* automatically enter Sleep Mode after a user inactivity timeout.
- During a datalogging session the Node will not respond to any commands, except the **Stop Node** command.
- If continuous datalogging is enabled, the Node will continue datalogging until it uses all available memory, it receives a **Stop Node** command, or it is powered off.

### Relevant Memory Locations

Node EEPROM 12: Active Channel Mask
Node EEPROM 14: Datalogging Sample Rate
Node EEPROM 16: Samples per Data Set
Node EEPROM 102: Continuous Datalogging Flag

*See also:* Data Format , Checksum, Download Page, Erase Session Data, Arm Node

## Download Page

When the Node is in datalogging mode, or when the Node is configured for “log only” or “log and transmit,” it stores the data to its internal memory. The **Download Page** command is used to retrieve a logged data session from the Node.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x05 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| [0x0 - 0xFFFF] | 2 | uint16 | Page Index to Download |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x05 | 1 | uint8 | Command ID |
| | 264 | uint16 | Data Points (132 2-byte values) |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Fail Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x05 | 1 | uint8 | Command ID |
| 0x21 | 2 | uint16 | Fail Response ID |

### Page Index

Each Node contains 2MB of memory. These 2MB are mapped to 8192 pages of data, each page containing 264 bytes.

Pages are numbered sequentially from 0 to 8191.

- Page 0 contains data points that represent the value in the Node’s EEPROM from locations 0 to 254.
- Page 1 contains data points that represent the value in the Node’s EEPROM from locations 256 to 510.
- Page 2 is the first page that will contain sampled data that was logged to the node.

### Data Sessions

The Node can contain multiple consecutive datalogging sessions. Each of these sessions has a leading multi-byte header which is used to identify the start of a new datalogging session and the end of the previous session. The header contains the datalogging information stored during particular session.

*See also:* Data Format , Checksum

### Session Header

The datalogging session header is a multiple-byte leading string indicating the start of the next session. The header can be found anywhere on a downloaded page and it can wrap between pages. The header can be in one of many formats:

*Header Format Version 1.0*

Header Format Version 1.0 has the following format:

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xFFFF | 2 | uint16 | Start of Header |
| 0xFD | 1 | uint8 | Header ID |
| [0x0 - 0x04] | 1 | uint8 | Trigger ID |
| 0x01 | 1 | uint8 | Header Version (Major) |
| 0x00 | 1 | uint8 | Header Version (Minor) |
| [0x0A - 0x3C] | 2 | uint16 | # of bytes before channel data |
| [0x64 - 0xFFDC] | 2 | uint16 | Samples per Data Set |
| [0x01 - 0xFFFF] | 2 | uint16 | Session Index |
| [0x00 - 0xFF] | 1 | uint8 | Active Channel Mask |
| [0x01 - 0xFF] | 2 | uint16 | Sample Rate |
| [0x0 - 0x32] | 2 | uint16 | # of User entered bytes |
| | 0 - 50 | any | User entered data (if any) |
| [0x0010] | 2 | uint16 | Bytes per Channel |
| [0x0 - 0xFF] | 1 | uint8 | Channel Equation ID (1st active ch) |
| [0x0 - 0xFF] | 1 | uint8 | Channel Unit ID (1st active ch) |
| [0x0 - 0xFFFFFFFF] | 4 | float | Channel Slope (1st active ch) |
| [0x0 - 0xFFFFFFFF] | 4 | float | Channel Offset (1st active ch) |
| Repeat Channel Action Information for all active channels (see Active Channel Mask) | | | |
| [0x08] | 2 | uint16 | # of bytes before end of header |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (seconds) |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (nanoseconds) |

*Channel Information*

*Header Format Version 2.0*

Header Format Version 2.0 has the following format (darker lines signify changes from 1.0):

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xFFFF | 2 | uint16 | Start of Header |
| 0xFD | 1 | uint8 | Header ID |
| [0x0 - 0x04] | 1 | uint8 | Trigger ID |
| 0x02 | 1 | uint8 | Header Version (Major) |
| 0x00 | 1 | uint8 | Header Version (Minor) |
| [0x0C - 0x3E] | 2 | uint16 | # of bytes before channel data |
| [0x64 - 0xFFDC] | 2 | uint16 | Samples per Data Set |
| [0x01 - 0xFFFF] | 2 | uint16 | Session Index |
| [0x00 - 0xFF] | 1 | uint8 | Active Channel Mask |
| [0x01 - 0xFF] | 2 | uint16 | Sample Rate |
| [0x01 - 0x03] | 1 | uint8 | Data Type |
| 0x00 | 1 | uint8 | (Unused byte) |
| [0x0 - 0x32] | 2 | uint16 | # of User entered bytes |
| | 0 - 50 | any | User entered data (if any) |
| [0x0010] | 2 | uint16 | Bytes per Channel |
| [0x0 - 0xFF] | 1 | uint8 | Channel Action Equation ID (1st active ch) |
| [0x0 - 0xFF] | 1 | uint8 | Channel Action Unit ID (1st active ch) |
| [0x0 - 0xFFFFFFFF] | 4 | float | Channel Action Slope (1st active ch) |
| [0x0 - 0xFFFFFFFF] | 4 | float | Channel Action Offset (1st active ch) |
| Repeat Channel Action Information for all active channels (see Active Channel Mask) | | | |
| [0x08] | 2 | uint16 | # of bytes before end of header |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (seconds) |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (nanoseconds) |

*Channel Action Information*

*Header Format Version 2.1*

Header Format Version 2.1 has the following format (darker lines signify changes from 2.0):

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xFFFF | 2 | uint16 | Start of Header |
| 0xFD | 1 | uint8 | Header ID |
| [0x0 - 0x04] | 1 | uint8 | Trigger ID |
| 0x02 | 1 | uint8 | Header Version (Major) |
| 0x01 | 1 | uint8 | Header Version (Minor) |
| [0x0C - 0x3E] | 2 | uint16 | # of bytes before channel data |
| [0x01 - 0xFFFF] | 2 | uint16 | Samples per Data Set ÷ 100 |
| [0x01 - 0xFFFF] | 2 | uint16 | Session Index |
| [0x00 - 0xFF] | 1 | uint8 | Active Channel Mask |
| [0x01 - 0xFF] | 2 | uint16 | Sample Rate |
| [0x01 - 0x03] | 1 | uint8 | Data Type |
| 0x00 | 1 | uint8 | (Unused byte) |
| [0x0 - 0x32] | 2 | uint16 | # of User entered bytes |
| | 0 - 50 | any | User entered data (if any) |
| [0x0010] | 2 | uint16 | Bytes per Channel |
| [0x0 - 0xFF] | 1 | uint8 | Channel Action Equation ID (1st active ch) |
| [0x0 - 0xFF] | 1 | uint8 | Channel Action Unit ID (1st active ch) |
| [0x0 - 0xFFFFFFFF] | 4 | float | Channel Action Slope (1st active ch) |
| [0x0 - 0xFFFFFFFF] | 4 | float | Channel Action Offset (1st active ch) |
| Repeat Channel Action Information for all active channels (see Active Channel Mask) | | | |
| [0x08] | 2 | uint16 | # of bytes before end of header |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (seconds) |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (nanoseconds) |

*Channel Action Information*

### Fixed Header
Each new session is always marked with a 0xFFFF (65535) at its start.

### Header Version
The header format that needs to be used can be determined by checking the header version bytes.

### Trigger ID
Each datalogging session has an ID signifying how the session was started:

| Trigger ID | Description |
|---|---|
| 0 | Started via a datalogging command |
| 1 | Ceiling Sensor Event Driven Trigger |
| 2 | Floor Sensor Event Driven Trigger |
| 3 | Ramp Up Sensor Event Driven Trigger |
| 4 | Ramp Down Sensor Event Driven Trigger |

### Samples per Data Set
The number of expected samples per data set. If the session was logged as a continuous datalogging session, this value is not applicable. If the session ended prematurely due to power failure or because the memory was completely filled, the value will not correspond to the actual number of samples. When parsing session data, the Samples per Data Set value should not be used to determine the end of the session. The end of the session should be determined by the start of the next session header. Header Versions 2.1 and above have the number of Samples per Data Set byte divided by 100 to allow for longer sessions. This value should be multiplied by 100 to obtain the true number of samples.

### Session Index
Each data logging session is stored with an index. The first session's index is 1 and subsequent sessions would have consecutive index numbers. When the logged sessions are erased the index will be reset, and the first data logging session after the erase will have a Session Index of 1.

### Active Channel Mask
The channel mask indicates the active channels that were logged in this session. Please see the *Channel Mask* section for more details.

### Sample Rate
The Sample Rate that was used during the datalogging session. See the Sample Rates section for a table of the values and their corresponding sampling rates.

### Data Type
Data type is the format which the node saves data to memory as. The following table shows the possible data values and their corresponding data types.

| Value | Type | Description |
|---|---|---|
| 1, 3 | uint16 | A/D value (bits), no conversions (slope & offset) applied |
| 2 | float | floating point value with any conversions (slope & offset) applied |

### User Entered Bytes
Up to 50 optional user entered bytes may have been sent to the node when triggering an **Armed Datalogging** session. The "**#** of user entered bytes" byte specifies how many user bytes follow in the packet. If the number of user entered bytes is an odd number, there will be 1 extra "buffer" byte appended to the user data that should be discarded.

### Channel Information
The Channel Information bytes show which calibration coefficients were enabled during datalogging for the active channels. The Equation ID and the Unit ID are each 1 byte values, while the Slope and Offset are 4 byte floating point values. For more information, please see the **Calibration Coefficients** section.

### Timestamp
The UTC timestamp represents the starting time of the logged session (the time applied to the very first data point). The nanoseconds should be appended to the seconds of the given timestamp. Incrementing the initial UTC timestamp by the sampling rate, one can determine the exact time of each stored data point.

### Session Data
During the actual datalogging session, the data from each active channel on the Node is written consecutively to the memory pages. For example, a Node with 3 active channels (CH1, CH3, CH4) would write data as CH1, CH3, CH4, CH1, CH3, CH4 and so forth. This data comes off the Node in the same pattern during download.

### Response Packet Checksum
This response packet contains a checksum of bytes 2 -265.

### Relevant Memory Locations
Node EEPROM 0: Current Log Page
Node EEPROM 2: Current Page Offset
Node EEPROM 4: Data Sets Stored

*See also:* Data Format, Checksum, Arm Node, Trigger Armed Datalogging, Erase Session Data, Calibration Coefficients, Sample Rates

## Erase Session Data

The **Erase Session Data** command is used to erase all sampled data stored on the Node's memory.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x06 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x08100CFF | 4 | uint32 | Command Bytes |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x06 | 1 | uint8 | Command ID |

**Fail Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x21 | 1 | uint8 | Fail Response ID |

### Response

The response is returned immediately when the erasing processes begins, and no acknowledgment is sent when the process completes. The process will take approximately 5 seconds, and the Node will not respond to any commands until the erasing is complete. Completion of the erase can be detected by repeated short pinging of the Node; when the erase is complete, the Node will come back on-line and respond successfully to the ping.

*See also:* Data Format, Trigger Armed Datalogging, Short Ping

## Initiate Real-Time Streaming

The **Initiate Real-Time Streaming** command is used to start a real-time streaming session on a Node. The Node will respond by immediately sending a stream of data packets as the sensors are read.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x38 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |

**Response:**
Parsing of the streaming packets should begin with the first 0xFF byte. The 0xFF byte is the header of the first valid data packet shown below as Byte 1. Each successive 0xFF indicates the start of a new data packet and the end of the previous packet.

**Response Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xFF | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Channel Value for first active channel |
| Repeat Channel Value bytes for each active channel | | | |
| [0x0 - 0xFF] | 1 | uint8 | Checksum |

*Checksum Range*

### Terminating Streaming

Continuous or Finite Streaming may be stopped at any time by issuing any byte to the Base Station. This will cause the Base Station to stop streaming. It will not however cause the Node to stop streaming. The Node will continue to stream until the set (finite) duration has elapsed, the power on the Node is cycled, or the **Stop Node** command has been issues to the Node.

### End of Stream

The normal end of a finite stream is marked by 4-6 consecutive 0xAA (170) bytes. When parsing the stream, these bytes should be used as a signal that finite streaming has ended.

### Response Packet Checksum

Unlike most checksums which are 2 bytes, the checksum of a Real-Time Streaming packet is a single byte.

### Relevant Memory Locations

Node EEPROM 16: Samples per Data Set
Node EEPROM 100: Continuous Streaming/LDC Flag

## Initiate Low Duty Cycle

The **Initiate Low Duty Cycle (LDC)** command is used to put the Node in LDC mode. The Node will send an LDC packet for every sample.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x05 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x02 | 1 | uint8 | Payload Length |
| 0x0038 | 2 | uint16 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Initial Response:**
An initial response comes directly from the Base Station to acknowledge that the command was received by the Base Station and sent to the Node.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Packet-Sent Acknowledgement |

See the Low Duty Cycle Data Packet section for details of the data packet that is sent from the node after successfully initiating Low Duty Cycle.

*See also:* Data Format (MSB, LSB), Checksum, Stop Node

## Initiate Synchronized Sampling

The **Synchronized Sampling** command is used to put the Node into Synchronized Sampling mode. Once in this mode, the node must receive a beacon from the Base Station to begin sampling and transmitting packets of data. These packets contain buffered, time-stamped data.

For Synchronized Sampling mode, the wireless sensor network needs to be configured so that each node is assigned an appropriate TDMA slot prior to issuing the Synchronized Sampling start command. To set up the wireless sensor network, use the Synchronized Sampling Wizard within the Node Commander software. Node Commander may also be used to start the network. Failure to configure the network correctly will lead to wireless packets colliding, resulting in data loss.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x05 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x02 | 1 | uint8 | Payload Length |
| 0x003B | 2 | uint16 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Initial Response:**
An initial response comes directly from the Base Station to acknowledge that the command was received by the Base Station and sent to the Node.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Packet-Sent Acknowledgement |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x07 | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x03 | 1 | uint8 | Payload Length |
| 0x003B | 2 | uint16 | Command ID |
| 0x00 | 1 | uint8 | Fail Type / Open Byte |
| [0x0 - 0xFF] | 1 | int8 | Node RSSI |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Fail Response:** No Response

See the Synchronized Sampling Data Packet section for details of the data packet that is sent from the node after successfully initiating Synchronized Sampling.

*See also:* Data Format, Checksum, Enable Beacon, Disable Beacon, Stop Node

## Read Single Sensor

The **Read Single Sensor** command is used to read the current value of a specified channel.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x03 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| 0x01 | 1 | uint8 | Command Byte |
| [0x01 - 0x08] | 1 | uint8 | Channel Number (1-8) |

**Success Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x03 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Value of the requested channel |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

**Fail Response:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x21 | 1 | uint8 | Fail Response ID |

### Channel Number

The Channel Number of the sensor to read. If a channel number is used that doesn't exist on the Node, erroneous response data will be returned.

Note: Values read using the **Read Single Sensor** command may be off due to there being no excitation time taken into account. The other full sampling modes use a Sampling Delay, which is an amount of time between sensor excitation power up and A/D sampling. This value can be configured using EEPROM 34.

### Relevant Memory Locations

Node EEPROM 34: Sampling Delay

*See also:* Data Format, Checksum

## Auto-Balance Channel

The **Auto-Balance Channel** command is used to auto-balance a particular channel on the Node. This command is only applicable to the differential channels on the V-Link, SG-Link and SG-Link OEM nodes.

**Command Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0x62 | 1 | uint8 | Command ID |
| [0x0 - 0xFFFF] | 2 | uint16 | Node Address |
| [0x01 - 0x08] | 1 | uint8 | Channel to balance (hardware specific, 1-8) |
| [0x0 - 0xFFFF] | 2 | uint16 | Target Balance Value |

**Success Response:** No Response
**Fail Response:** No Response

### Channel to Balance

Channels 1, 2, 3 and/or 4 on the V-Link.
Channel 1 on the SG-Link and SG-Link OEM.

### Target Balance Value

The target balance value represents the desired sensor output value in bits, with a range of 0-4096. All differential inputs have a programmable offset feature that allows the user to trim sensor offset (see hardware user manual for more information.) This programmable offset can be manually altered by writing to Node EEPROM locations 26-32, or auto-tuned such that the sensor output is balanced to a user-defined target. For example, a common use is to auto-balance to mid-scale (2048 bits) to obtain maximum bipolar dynamic range.

### Relevant Memory Locations

Node EEPROM 26: PGA Offset 1
Node EEPROM 28: PGA Offset 2
Node EEPROM 30: PGA Offset 3
Node EEPROM 32: PGA Offset 4

*See also:* Data Format

# Data Packets

## Low Duty Cycle Packet

These packets are received at various, defined intervals after an **Initiate Low Duty Cycle** command has been sent to a node that is *not* configured to send Buffered packets (default configuration).

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x07 | 1 | uint8 | Delivery Stop Flag |
| 0x04 | 1 | uint8 | App Data Type |
| [0x1 - 0xFFFE] | 2 | uint16 | Node Address |
| [0x2 - 0x6A] | 1 | uint8 | Payload Length |
| 0x02 | 1 | uint8 | App ID |
| [0x0 - 0xFF] | 1 | uint8 | Channel Mask |
| [0x1 - 0x7B] | 1 | uint8 | Sample Rate |
| [0x1 - 0x4] | 1 | uint8 | Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Timer Tick |
| [0x0 - 0xFFFFFFFF] | 2 - 4 | uint16, uint32, float | Channel Data for first active channel |
| Repeat Channel Data bytes for each active channel | | | |
| [0x0 - 0xFF] | 1 | int8 | RESERVED |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

### Channel Mask

The Channel Mask is a bitwise representation of which channels are active and have data in the packet.

### Sample Rate

The Sample Rate that was used during the sampling session. See the Sample Rates section for a table of the values and their corresponding sampling rates.

### Data Type

The Data Type byte signifies the type of Channel Data that is contained in the packet, and can be any of the following values:

- 0x01 = 2 byte unsigned integer (uint16) (bit-shifted)
- 0x02 = 4 byte float (float)
- 0x03 = 2 byte unsigned integer (uint16)
- 0x04 = 4 byte unsigned integer (uint32)

*Note: A data type of 0x1 signifies a 2 byte integer that must be divided by 2 to get the correct value.*

### Timer Tick

The Tick represents a counter for the number of packets sent. Once this value reaches 65535, it rolls over to 0.

### Relevant Memory Locations

Node EEPROM 12: Active Channel Mask
Node EEPROM 16: Samples Per Data Set
Node EEPROM 72: LDC / Synchronized Sampling Rate

*See also:* Data Format, Channel Mask, Checksum

## Buffered Low Duty Cycle Packet

These packets are received at various, defined intervals after an **Initiate Low Duty Cycle** command has been sent to a node that is configured to send Buffered LDC packets.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x07 | 1 | uint8 | Delivery Stop Flag |
| 0x0D | 1 | uint8 | App Data Type |
| [0x01 - 0xFFFE] | 2 | uint16 | Node Address |
| [0x02 - 0x6A] | 1 | uint8 | Payload Length |
| 0x02 | 1 | uint8 | App ID |
| [0x0 - 0xFF] | 1 | uint8 | Channel Mask |
| [0x1 - 0x7B] | 1 | uint8 | Sample Rate |
| [0x1 - 0x4] | 1 | uint8 | Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Sweep Tick |
| [0x0 - 0xFFFFFFFF] | 2 - 4 | uint16, uint32, float | Channel Data, for first active channel, for first Sweep |
| Repeat Channel Data bytes for each active channel, and for each Sweep in the packet | | | |
| [0x0 - 0xFF] | 1 | int8 | Node RSSI |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

### Channel Mask

The Channel Mask is a bitwise representation of which channels are active and have data in the packet.

### Sample Rate

The Sample Rate that was used during the sampling session. See the Sample Rates section for a table of the values and their corresponding sampling rates.

### Data Type

The Data Type byte signifies the type of Channel Data that is contained in the packet, and can be any of the following values:

- 0x01 = 2 byte unsigned integer (uint16) (bit-shifted)
- 0x02 = 4 byte float (float)
- 0x03 = 2 byte unsigned integer (uint16)
- 0x04 = 4 byte unsigned integer (uint32)

*Note: A data type of 0x1 signifies a 2 byte integer that must be divided by 2 to get the correct value.*

### Timer Tick

The Tick represents a counter for the number of packets sent. Once this value reaches 65535, it rolls over to 0. Although each packet may contain more than one sweep per channel, there is only one tick acquired per packet. Therefore the tick value must be incremented manually for each sweep within the packet.

### Relevant Memory Locations

Node EEPROM 12: Active Channel Mask
Node EEPROM 16: Samples Per Data Set
Node EEPROM 72: LDC / Synchronized Sampling Rate

*See also:* Data Format, Channel Mask, Checksum

## Synchronized Sampling Packet

These packets are received at various, defined intervals after an **Initiate Synchronized Sampling** command has been sent to an LXRS node.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x07 | 1 | uint8 | Delivery Stop Flag |
| 0x0A | 1 | uint8 | App Data Type |
| [0x01 - 0xFFFE] | 2 | uint16 | Node Address |
| [0x02 - 0x6A] | 1 | uint8 | Payload Length |
| [0x1 - 0x2] | 1 | uint8 | Sample Mode |
| [0x0 - 0xFF] | 1 | uint8 | Channel Mask |
| [0x1 - 0x7B] | 1 | uint8 | Sample Rate |
| [0x1 - 0x4] | 1 | uint8 | Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Sweep Tick |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (seconds) |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (nanoseconds) |
| [0x0 - 0xFFFFFFFF] | 2 - 4 | uint16, uint32, float | Channel Data, for first active channel, for first Sweep |
| Repeat Channel Data bytes for each active channel, and for each Sweep in the packet | | | |
| [0x0 - 0xFF] | 1 | int8 | Node RSSI |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

### Sample Mode

The response packet's Sample Mode can contain the following values for their current mode of operation:

- 0x01 = Burst Mode
- 0x02 = Continuous Mode

### Channel Mask

The Channel Mask is a bitwise representation of which channels are active and have data in the packet.

### Sample Rate

The Sample Rate that was used during the sampling session. See the Sample Rates section for a table of the values and their corresponding sampling rates.

### Data Type

The response packet data type can contain the following values for the type of data transmitted:

- 0x01 = 2 byte unsigned integer (uint16) (bit-shifted)
- 0x02 = 4 byte float (float)
- 0x03 = 2 byte unsigned integer (uint16)
- 0x04 = 4 byte unsigned integer (uint32)

*Note: A data type of 0x01 signifies a 2 byte integer that must be divided by 2 to get the correct value.*

### Timestamp/Tick

Although each packet may contain more than one sweep per channel, there is only one timestamp and tick acquired per packet. Therefore the tick and timestamp values must be incremented manually for each sweep within the packet. This can be done via the following:

- Using the sample rate of the node, determine your increment value:
  - If sample rate is 32 Hz, the increment is 0.03125. (1 / 32)
  - If sample rate is 1 per 2 seconds, the increment is 2.
  - Multiply the increment by 1,000,000,000 to convert to nanoseconds.
- For each sweep in the packet, add the calculated value to the original timestamp.
- For each sweep, increment the tick by 1.

### Channel Data

Multiple sweeps of data per channel may be contained in one packet. The number of sweeps in the packet can be determined via the following:

- **Bytes Before Data** = 14 (Bytes 7 – 20 are not channel data)
- **Bytes Per Sample** = 2 for 2 byte data, or 4 for float data (determined using Byte 10 – Data Type)
- **#Sweeps** = ((Payload Length – **Bytes Before Data**) / # of Active Channels) / **Bytes Per Sample**)

Channel data should be parsed in the following manner:

-Sweep 1, First active channel
-Sweep 1, Next active channel (Repeat for each active channel)
-Sweep 2, First active channel
-Sweep 2, Next active channel (Repeat for each active channel)
-Repeat for each Sweep in the packet

### Relevant Memory Locations

Node EEPROM 12: Active Channel Mask
Node EEPROM 16: Samples Per Data Set
Node EEPROM 72: LDC / Synchronized Sampling Rate

*See also:* Data Format, Channel Mask, Checksum, Enable Beacon, Disable Beacon, Stop Node

## Diagnostic Packet*

The **Diagnostic Packet** may be received at various (low rate) intervals from an LXRS node, and gives information on that Node.

*Note: This packet is currently only supported by select nodes and firmware.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x07 | 1 | uint8 | Delivery Stop Flag |
| 0x11 | 1 | uint8 | App Data Type |
| [0x01 - 0xFFFE] | 2 | uint16 | Node Address |
| [0x02 - 0x6A] | 1 | uint8 | Payload Length |
| [0x0 - 0xFF] | 1 | uint8 | Diagnostic Packet Interval |
| [0x0 - 0xFFFF] | 2 | uint16 | Tick |
| [0x0 - 0xFF] | 1 | uint8 | Info Item 1 Length |
| [0x01 - 0x03] | 1 | uint8 | Info Item 1 ID |
| [0x0 - 0xFFFFFFFF] | 1 - 4 | | Info Item 1 Value |
| Repeat Info Item *Length, ID,* and *Value* for all the Info Items in the packet | | | |
| [0x0 - 0xFF] | 1 | int8 | Node RSSI |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

### Diagnostic Packet Interval

The interval of which the Diagnostic Packet is transmitted. The 2 most significant bits of this byte signify the interval type (seconds, minutes, or hours). The remaining 6 bits are the interval value. The interval types are as follows (shown in binary):

- 00 = Seconds
- 01 = Minutes
- 10 = Hours

Example (in bits):

01 10 1011 = 43 minutes

### Tick

The Tick represents a counter for the Diagnostic Packet, signifying how many of these packets have been sent. When this value reaches the max 2-byte value (0xFFFF), it will roll over to 0.

### Info Items

An **Info Item** consists of Length, ID, and Value bytes. Each Info Item can be parsed by first looking at the length byte to determine how many bytes make up the rest of the rest of the Info Item (ID and Value). The **Info Item ID** can then be compared against the table below to determine how to parse the value bytes that follow. Multiple Info Items can be in one packet. Use the payload length to determine the number of Info Items in the packet.

### Info Item Length

The length (# of bytes) of the Info Item, including the **Info Item ID** and **Info Item Value** bytes. This length value does *not* include the **Info Item Length** byte itself.

### Info Item ID / Value

The **Info Item ID** represents the type of information that is given in the next **Info Item Value** bytes.

**Info Item IDs**:

| ID | Description | Data Values | #Bytes | Type | Unit |
|---|---|---|---|---|---|
| 0x01 | Transmit Info | Total Transmissions | 4 | uint32 | counts |
| | | Total Retransmissions | 4 | uint32 | counts |
| | | Total Dropped Packets | 2 | uint16 | counts |
| 0x02 | Active Running Time | - | 4 | uint32 | seconds |
| 0x03 | Battery Life Remaining | - | 1 | uint8 | percent |

- **(0x01) Transmit Info** - Gives details of Transmission info, including Total Transmissions, Total Retransmissions, and Total Dropped Packets
  - **Total Transmissions** - # of unique packets transmitted (not including retransmissions)
  - **Total Retransmissions** - # of retransmitted packets (packets are retransmitted when a node doesn't receive an acknowledgement from the base station)
  - **Total Dropped Packets** - # of packets node has discarded due to buffer overflow or exceeding the max # of retransmissions per packet
- **(0x02) Active Running Time** - # of seconds the node has been in a sampling mode
- **(0x03) Battery Life Remaining** - % estimated battery level (0% to 100%)

Full Example:
If the Info Item Length byte is 0x0B, the next 11 bytes make up the Info Item ID and Value. If the next byte is 0x01, then we know the Info Item Value is signifying *Transmit Info* which is made up of *Total Transmissions, Total Retransmissions,* and *Total Dropped Packets*. From the table we know to parse the next 4 bytes as a uint32 representing the Total Transmissions, the next 4 bytes as a uint32 representing the Total Retransmissions, and the last 2 bytes as a uint16 representing the Total Dropped Packets.

*See also:* Data Format, Checksum

## Asynchronous Digital-Only Packet*

The **Asynchronous Digital-Only** data packet contains time and digital state values collected when a node was triggered by digital pulses.

*Note: This packet is currently only supported by select nodes and firmware.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x07 | 1 | uint8 | Delivery Stop Flag |
| 0x0E | 1 | uint8 | App Data Type |
| [0x01 - 0xFFFE] | 2 | uint16 | Node Address |
| [0x02 - 0x6A] | 1 | uint8 | Payload Length |
| [0x0 - 0xFFFF] | 2 | uint16 | Digital Channel Mask |
| [0x0 - 0xFFFF] | 2 | uint16 | Event Tick |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (seconds) |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (nanoseconds) |
| [0x0 - 0xFFFF] | 2 | uint16 | Timestamp Offset - Event 1 |
| [0x0 - 0xFFFF] | 2 | uint16 | Digital Channel Data - Event 1 |
| Repeat Timestamp and Digital Channel bytes for all Events | | | |
| [0x0 - 0xFF] | 1 | int8 | Node RSSI |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

### Digital Channel Mask

The Digital Channel Mask represents the digital channels that are actively being monitored by the node.

### Event Tick

The Event Tick represents a counter for each event. Although multiple events may be in the same packet, only one Event Tick will be found per packet. The tick for each event can be found by incrementing the Event Tick for each event that is contained in that packet.

### Timestamp

The UTC Timestamp bytes represent the initial timestamp in which each Event's Timestamp Offset must be added to in order to calculate the UTC Timestamp for each Event.

### Events

Each event contained in the packet is made up of a **2-byte Timestamp Offset** and **2 bytes of Digital Channel Data**. The Timestamp Offset must be divided by 32,768 to get its time in seconds. Each bit in the digital channel data represents a digital line. However, the Digital Channel Mask should be interrogated to determine which of these lines have a valid value.

Multiple events may be contained in one packet. The number of events in the packet can be determined via the following: **#Events** = ((Payload Length - 12) / 4)
Each event has a timestamp offset that much be added to the packet's absolute timestamp. This can be done via the following:

EventTimestamp = PacketTimestamp + (EventTimestampOffset / 32,768)

## Asynchronous Digital & Analog Packet*

The **Asynchronous Digital & Analog** data packet contains time, digital state, and analog sensor values collected when a node was triggered by digital pulses.

*Note: This packet is currently only supported by select nodes and firmware.

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| 0x07 | 1 | uint8 | Delivery Stop Flag |
| 0x0F | 1 | uint8 | App Data Type |
| [0x01 - 0xFFFE] | 2 | uint16 | Node Address |
| [0x02 - 0x6A] | 1 | uint8 | Payload Length |
| [0x0 - 0xFFFF] | 2 | uint16 | Channel Mask |
| [0x02 - 0x04] | 1 | uint8 | Data Type |
| [0x0 - 0xFFFF] | 2 | uint16 | Event Tick |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (seconds) |
| [0x0 - 0xFFFFFFFF] | 4 | uint32 | UTC Timestamp (nanoseconds) |
| [0x0 - 0xFFFF] | 2 | uint16 | Timestamp Offset - Event 1 |
| [0x0 - 0xFFFF] | 2 | uint16 | Digital Channel Data - Event 1 |
| [0x0 - 0xFFFFFFFF] | 2 or 4 | uint16, uint32, float | Analog Channel Data Ch1 - Event 1 |
| Repeat Analog Channel Data for each digital channel that is active (high) in the Digital Channel Data bytes for the current Event. | | | |
| Repeat Timestamp, Digital Channel Data, and Analog Channel Data bytes for all events | | | |
| [0x0 - 0xFF] | 1 | int8 | Node RSSI |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

### Channel Mask

The Channel Mask represents the digital channels that are actively being monitored by the node.

### Data Type

The Data Type determines the format of *Analog* data that is transmitted in the packet. It can contain the following value:

- 0x02 = 4 byte float (float)
- 0x03 = 2 byte unsigned integer (uint16)
- 0x04 = 4 byte unsigned integer (uint32)

Analog data should be parsed accordingly. For example, if the Data Type is 0x02 (float), there will be 4 bytes making up the Analog data. If the Data Type is 0x03 (uint16), there will be 2 bytes making up the Analog data.

### Event Tick
The Event Tick represents a counter for each event. Although multiple events may be in the same packet, only one Event Tick will be found per packet. The tick for each event can be found by incrementing the Event Tick for each event that is contained in that packet.

### Timestamp
The UTC Timestamp bytes represent the initial timestamp in which each Event's Timestamp Offset must be added to in order to calculate the UTC Timestamp for each Event.

### Events
Each event contained in the packet is made up of a **2-byte timestamp offset**, **2-bytes of digital channel data**, and **X bytes of analog channel data**. The Timestamp Offset must be divided by 32,768 to get its time in seconds. Each bit in the digital channel data represents a digital line. However, the Channel Mask should be interrogated to determine which of these lines have a valid value. The analog channel data will only be present when the digital channel line is active (a value of 1). Note that the analog channel data needs to be read off in the specific data type that is denoted by the Data Type byte.

- Ex. The first event in the packet contains a digital value of 13, which is 0000 0000 0000 1101 in binary, meaning digital lines 1, 3, and 4 are active. The Data Type of the packet is 0x02, meaning the analog data points are 4-byte floating point values. So the next 12 bytes (3 channels * 4 bytes per ch) in the packet will be the analog values for channels 1, 3, and 4.

Each packet may contain more than one event. Each event has a timestamp offset that must be added to the packet's absolute timestamp. This can be done via the following:

EventTimestamp = PacketTimestamp + (EventTimestampOffset / 32,768)

# Other Details

## Sample Rates

### Synchronized Sampling / Low Duty Cycle Sample Rates

The table below lists the data values and their corresponding Sample Rates that apply to the **Synchronized Sampling** and **Low Duty Cycle** sampling modes.

| Value | Sample Rate |
|---|---|
| 123 | 1 sample per 60 minutes |
| 122 | 1 sample per 30 minute |
| 121 | 1 sample per 10 minutes |
| 120 | 1 sample per 5 minutes |
| 119 | 1 sample per 2 minutes |
| 118 | 1 sample per 60 seconds |
| 117 | 1 sample per 30 seconds |
| 116 | 1 sample per 10 seconds |
| 115 | 1 sample per 5 seconds |
| 114 | 1 sample per 2 seconds |
| 113 | 1 Hz |
| 112 | 2 Hz |
| 111 | 4 Hz |
| 110 | 8 Hz |
| 109 | 16 Hz |
| 108 | 32 Hz |
| 107 | 64 Hz |
| 106 | 128 Hz |
| 105 | 256 Hz |
| 104 | 512 Hz |
| 103 | 1024 Hz |
| 102 | 2048 Hz |
| 101 | 4096 Hz |

### Armed Datalogging Sample Rates

The table below lists the data values and their corresponding Sample Rates that apply to the **Armed Datalogging** sampling mode.

| Value | Rate |
|---|---|
| 1 | 2048 Hz |
| 2 | 1024 Hz |
| 3 | 512 Hz |
| 4 | 256 Hz |
| 5 | 128 Hz |
| 6 | 64 Hz |
| 7 | 32 Hz |

## Broadcast Address

A special node address referred to as the *Broadcast Address* exists. This Broadcast Address is 65535 or 0xFFFF. Nodes shipped from the factory and any nodes in use by the customer should not be addressed as 65535; Node addresses should always range between 1 and 65534.

When Nodes are idling and listening for commands, they are waiting for commands directed at them specifically. For example, in the case of the Short Ping command, we send 3 bytes, being an initial command byte 0x02 followed by two bytes representing the address of a unique Node. This method insures that only the target Node responds to the command.

However, in many cases we want to command many Nodes to do the same thing. To handle this situation, all Nodes are always listening for commands to the Broadcast Address 65535 in addition to commands to their unique address. As an example, this special feature allows us to command many Nodes to trigger an Armed Datalogging session all at the same time. This is useful to generate sensor data that has the same session start time.

This Broadcast Address can most obviously be used in the following commands:

- Initiate Low Duty Cycle
- Initiate Synchronized Sampling
- Trigger Armed Datalogging
- Erase Session Data
- Write Node EEPROM
- Initiate Sleep Mode
- Stop Node

In addition, commands for waking multiple Nodes, cycling the power on multiple Nodes, and finding Nodes with unknown addresses can all be fabricated with this special Broadcast Address.

## Node Discovery

On power up, the Node will transmit two identification packets. The packets are sent out on all 16 radio frequencies, allowing any Base Station within range to receive the identification packets, regardless of the Base Station's current frequency assignment. The Base Station immediately passes these packets to the host serial port.
The Node will only send out a Node discovery packet when its power is cycled.

**Node Discovery Packet:**

| Value | # Bytes | Type | Description |
|---|---|---|---|
| 0xAA | 1 | uint8 | Start of Packet |
| [0x0 - 0xFF] | 1 | uint8 | Delivery Stop Flag |
| 0x00 | 1 | uint8 | App Data Type |
| [0x1 - 0xFFFE] | 2 | uint16 | Node Address |
| 0x03 | 1 | uint8 | Payload Length |
| [0x0 - 0xFF] | 1 | uint8 | Radio Channel (Frequency) |
| [0x0 - 0xFFFF] | 2 | uint16 | Model Number |
| [0x0 - 0xFF] | 1 | int8 | RESERVED |
| [0x0 - 0xFF] | 1 | int8 | Base Station RSSI |
| [0x0 - 0xFFFF] | 2 | uint16 | Checksum |

*Checksum Range*

### Delivery Stop Flag

The Delivery Stop Flag byte for the Node Discovery Packet must be checked to ensure that bit 4 (0000 0000) is a 0. This can be achieved by ANDing (&) the byte with 0x08 and checking the result is 0.

### Radio Channel (Frequency)

The Radio Channel value corresponds to the channel on which the Node is currently communicating. This is the same value stored in Node EEPROM 90 and sends only values that are valid within EEPROM 90. Please, reference the Node EEPROM map for position 90 to identify valid radio channels.

### Model Number

The Model Number corresponds to the model number of the Node stored in Node EEPROM 112. This value identifies the type of Node that had its power cycled. Please reference the Node EEPROM map for position 112 to identify valid model numbers.

*See also:* Data Format, Checksum

## Calibration Coefficients

Calibration coefficients are linear scaling filters applied to convert a channel's raw bit value to physical or engineering units. Calibration coefficients can be generated through Node Commander calibration wizards or via manual calibration by the end user. Please see the Node hardware manual for more information on general calibration procedures. This documentation explains how to use existing calibration coefficients in custom software.

Please note:

1) The Calibration coefficient offset is different from a node's hardware offset settings.
2) When the Node is transmitting 2-byte integer values, Calibration coefficients represent a post-processing step. The conversion from bits to physical units takes place on the host computer and not on the Node. However, when the Node is transmitting 4-byte float values, the conversion *will* occur on the Node and the received data values will already have been converted.

Calibration coefficient information is stored in non-volatile EEPROM memory on the node, at locations 150 through 228 inclusive. Each channel requires 10 bytes to hold the equation ID, unit ID, slope, and offset values:

| Description | # Bytes | Type |
|---|---|---|
| Equation ID | 1 | uint8 |
| Unit ID | 1 | uint8 |
| Slope | 4 | float |
| Offset | 4 | float |

The EEPROM Locations of Calibration Coefficients for each specific channel are as follows:

| Channel | EEPROM Address Range |
|---|---|
| Ch1 | 150 - 158 |
| Ch2 | 160 - 168 |
| Ch3 | 170 - 178 |
| Ch4 | 180 - 188 |
| Ch5 | 190 - 198 |
| Ch6 | 200 - 208 |
| Ch7 | 210 - 218 |
| Ch8 | 220 - 228 |

Below are more in depth explanations of each calibration coefficient field.

### Equation ID

The Equation ID is an integer value used to determine which calibration coefficient is applied to the current channel. The Equation ID and Unit ID occupy the same EEPROM location. The Equation ID is the MSB and the Unit ID is the LSB of this location.

The Standard Format equation is used by LORD for all new devices. The Legacy Strain and Legacy Acceleration formulas were used by previous MicroStrain software and may be present on older nodes. The Legacy equations each imply a unit type, microstrain (με) for Legacy Strain and G's for Legacy Acceleration, while the standard equation is more flexible and doesn't enforce a specific unit. With the Standard Equation, the unit is stored separately. We recommend using the Standard Format for all custom calibration. The table below shows the three equations.

| Equation ID | Name | Equation |
|---|---|---|
| 0x00 | Bits (None) | `value = bits` |
| 0x01 | Legacy Strain | `value = slope * (bits + offset)` |
| 0x02 | Legacy Acceleration | `value = (bits – offset) / slope` |
| 0x04 | Standard Format | `value = slope * bits + offset` |

Any other value for the Equation ID will be treated as if no calibration coefficient is applied to the channel.

### Unit Type ID

The Unit Type ID is an integer value used to determine which unit is applied to the current channel. The Unit Type ID value is the LSB of EEPROM 0. The table below shows the Unit Type ID values and their corresponding unit symbols and types.

| Unit Type ID | Unit Symbol | Unit Type |
|---|---|---|
| 0x00 | | other |
| 0x01 | bits | bits |
| 0x02 | ε | Strain |
| 0x03 | με | microStrain |
| 0x04 | G | acceleration due to gravity |
| 0x05 | m/s² | meters per second squared |
| 0x06 | V | volts |
| 0x07 | mV | millivolts |
| 0x08 | µV | microvolts |
| 0x09 | °C | degrees Celsius |
| 0x0A | K | Kelvin |
| 0x0B | °F | degrees Fahrenheit |
| 0x0C | m | meters |
| 0x0D | mm | millimeters |
| 0x0E | µm | micrometers |
| 0x0F | Lbf | pound force |
| 0x10 | N | Newtons |
| 0x11 | kN | kiloNewtons |
| 0x12 | kg | kilograms |
| 0x13 | bar | bar |
| 0x14 | psi | pounds per square inch |
| 0x15 | atm | atmospheric pressure |
| 0x16 | mmHg | millimeters of mercury |
| 0x17 | Pa | Pascal |
| 0x18 | MPa | megaPascal |
| 0x19 | kPa | kiloPascal |
| 0x1A | degrees | degrees |
| 0x1B | degrees/s | degrees per second |
| 0x1C | rad/s | radians per second |
| 0x1D | % | percent |
| 0x1E | rpm | revolutions per minute |
| 0x1F | Hz | hertz |
| 0x20 | %RH | relative humidity |
| 0x21 | mV/V | milliVolt/Volt |

### Slope and Offset

The slope and offset are 4-byte floating point values in IEE 754-single precision format. The example below demonstrates how to read the floating point values in C++:

```
typedef unsigned char BYTE;
typedef unsigned short WORD;

float MakeFloat(WORD eepromLoWord, WORD eepromHiWord)
{
	float f = 0;

	//map the float to a byte array
	BYTE* tmp = (BYTE*)&f;

	//set each byte of the float via the tmp byte pointer
	tmp[0] = eepromLoWord >> 8;
	tmp[1] = eepromLoWord & 0xFF;
	tmp[2] = eepromHiWord >> 8;
	tmp[3] = eepromHiWord & 0xFF;

	return f;
}
```

The following table displays the standard slope and offset values to convert single-ended input channels to volts, and the internal temperature sensor to degrees Celsius.

| Unit | Slope | Offset | Applies To | |
|---|---|---|---|---|
| | | | Node Type | Channels |
| Volts | .000732 | 0 | V-Link | 5, 6, 7 |
| | | | SG-Link | 4 |
| | | | SG-Link OEM | 4 |
| Temp C | .117188 | -67.84 | V-Link | 8 |
| | | | G-Link | 4 |
| | | | SG-Link | 3 |
| | | | SG-Link OEM | 3 |

### Example

Below is an example of converting EEPROM addresses to their corresponding calibration coefficient. The base address is already included in the EEPROM address.

| EEPROM Address | Value (Dec) | Value (Hex) | |
|---|---|---|---|
| 180 | 1033 | 0x04 | → *Equation ID* |
| | | 0x09 | → *Unit Type ID* |
| 182 | 17152 | 0x43 | *Slope* |
| | | 0x00 | |
| 184 | 61501 | 0xF0 | |
| | | 0x3D | |
| 186 | 5294 | 0x14 | *Offset* |
| | | 0xAE | |
| 188 | 34754 | 0x87 | |
| | | 0xC2 | |

The above EEPROM values correspond to channel 4. This can be told because the EEPROM address, EEPROM 180, corresponds to channel 4 in the EEPROM map.

First determine which calibration coefficient is to be applied to channel 4. The calibration coefficient Equation ID is the MSB of EEPROM 180, which is 0x04. The Equation ID therefore corresponds to the Standard Format. Thus, the formula to use is slope * bits + offset as described earlier.

Second, the Unit Type ID can be calculated. The calibration coefficient Unit Type ID is the LSB of EEPROM 180, which is 0x09. The Unit Type ID therefore corresponds to degrees Celsius.

Next, the slope and offset must be calculated. The slope has the values 0x4300 and 0xF03D for EEPROM addresses 182 and 184, respectively. Using the formula written earlier and knowing these 4 bytes are in big endian format, the floating point equivalent is 0.117188.

The offset has the values 0x14AE and 0x87C2 for EEPROM addresses 186 and 188, respectively. Using the same formula as calculating the slope and knowing these 4 bytes are in big endian format, the floating point equivalent is -67.84.

As noted in the table of slopes and offsets for the Standard Format, this calibration coefficient converts bits to degrees Celsius for channel 4.

## Channel Mask

The Channel Mask is a value contained in EEPROM address 12 of a Node which dictates which of the Node's channels are active. This value sets the channels that will be sampled during any type of data sampling mode. The mask is an 8-bit value. Each of the 8 bits of the mask correspond to one of the Node's channels. Bit 1 corresponds to channel 1, bit 2 to channel 2, bit 8 to channel 8, etc. When the bit is set to 1, the session will sample data for that channel. If the bit is set to 0, the session will not sample data for that channel.

**Mask Layout**

| Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 |
|---|---|---|---|---|---|---|---|
| CH8 | CH7 | CH6 | CH5 | CH4 | CH3 | CH2 | CH1 |

**Example showing channels 1, 3 and 4 active**

| Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

In this example the mask has been set with channels 1, 3 and 4 active (they are set with 1's). This value equates as follows:

- Binary = 1101
- Decimal = 13
- Hex = 0x0D

### C++ Example Functions:

```
typedef unsigned char BYTE;

BYTE ToMask(bool ch1,bool ch2,bool ch3,bool ch4,bool ch5,bool ch6,bool ch7,bool ch8)
{
    BYTE channelMask = 0;
    if(ch1){ channelMask |= (1 << 0);} //channel 1 - 0000 0001
    if(ch2){ channelMask |= (1 << 1);} //channel 2 - 0000 0010
    if(ch3){ channelMask |= (1 << 2);} //channel 3 - 0000 0100
    if(ch4){ channelMask |= (1 << 3);} //channel 4 - 0000 1000
    if(ch5){ channelMask |= (1 << 4);} //channel 5 - 0001 0000
    if(ch6){ channelMask |= (1 << 5);} //channel 6 - 0010 0000
    if(ch7){ channelMask |= (1 << 6);} //channel 7 - 0100 0000
    if(ch8){ channelMask |= (1 << 7);} //channel 8 - 1000 0000
    return channelMask;
}

void FromMask(BYTE channelMask)
{
    bool ch1 = (channelMask & (1 << 0)) != 0; //channel 1 - 0000 0001
    bool ch2 = (channelMask & (1 << 1)) != 0; //channel 2 - 0000 0010
    bool ch3 = (channelMask & (1 << 2)) != 0; //channel 3 - 0000 0100
    bool ch4 = (channelMask & (1 << 3)) != 0; //channel 4 - 0000 1000
    bool ch5 = (channelMask & (1 << 4)) != 0; //channel 5 - 0001 0000
    bool ch6 = (channelMask & (1 << 5)) != 0; //channel 6 - 0010 0000
    bool ch7 = (channelMask & (1 << 6)) != 0; //channel 7 - 0100 0000
    bool ch8 = (channelMask & (1 << 7)) != 0; //channel 8 - 1000 0000
}

void SetChannelInMask(int channel,bool enable,BYTE& channelMask)
{
    //channel 1 is bit 0
    channel -= 1;
    if(enable)
    {
        //Adding a channel and 'or' in the channel's bit.
        channelMask |= (1<<channel);
    }
    else
    {
        //Removing a channel, mask out the channel's bit.
        channelMask &= 0xff ^(1<<channel);
    }
}

bool IsChannelSetInMask(int channel,BYTE& channelMask)
{
    //Channel 1 is bit 0
    channel-=1;
    //Check the channel's specific bit
    return (channelMask & (1<<channel)) != 0;
}
```

## CP210x USB to UART Bridge Controller

The LORD WSDA-Base-104 USB and WSDA-Base-101 Analog Base Stations operate with a special driver installed on the operating system. This driver creates a Virtual COM Port that can be communicated with just like any other serial port. While normally installed during installation of LORD's Node Commander software, the driver may also be downloaded from Silicon Labs at: https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx

With the installation of this driver, the developer will find that communication between host and base station will be, for all intents and purposes, typical serial communication. The various coding languages will interact as if a standard serial port existed.

To determine if the driver is properly installed, connect the base station to the host and follow these steps:

1. Click Start on the Windows desktop.
2. Click Control Panel.
3. Click Device Manager.
4. Device Manger window appears.
5. Locate *Ports(Com & LPT)* in the tree.
6. Expand the *Ports* tree.
7. Do you see a device named 'CP210x USB to UART Bridge Controller (ComX)' or 'Silicon Labs CP210x USB to UART Bridge (ComX)? (with X representing a comm port number)
8. If so, the driver is installed.
9. If not, the base station is malfunctioning or the driver is not installed.

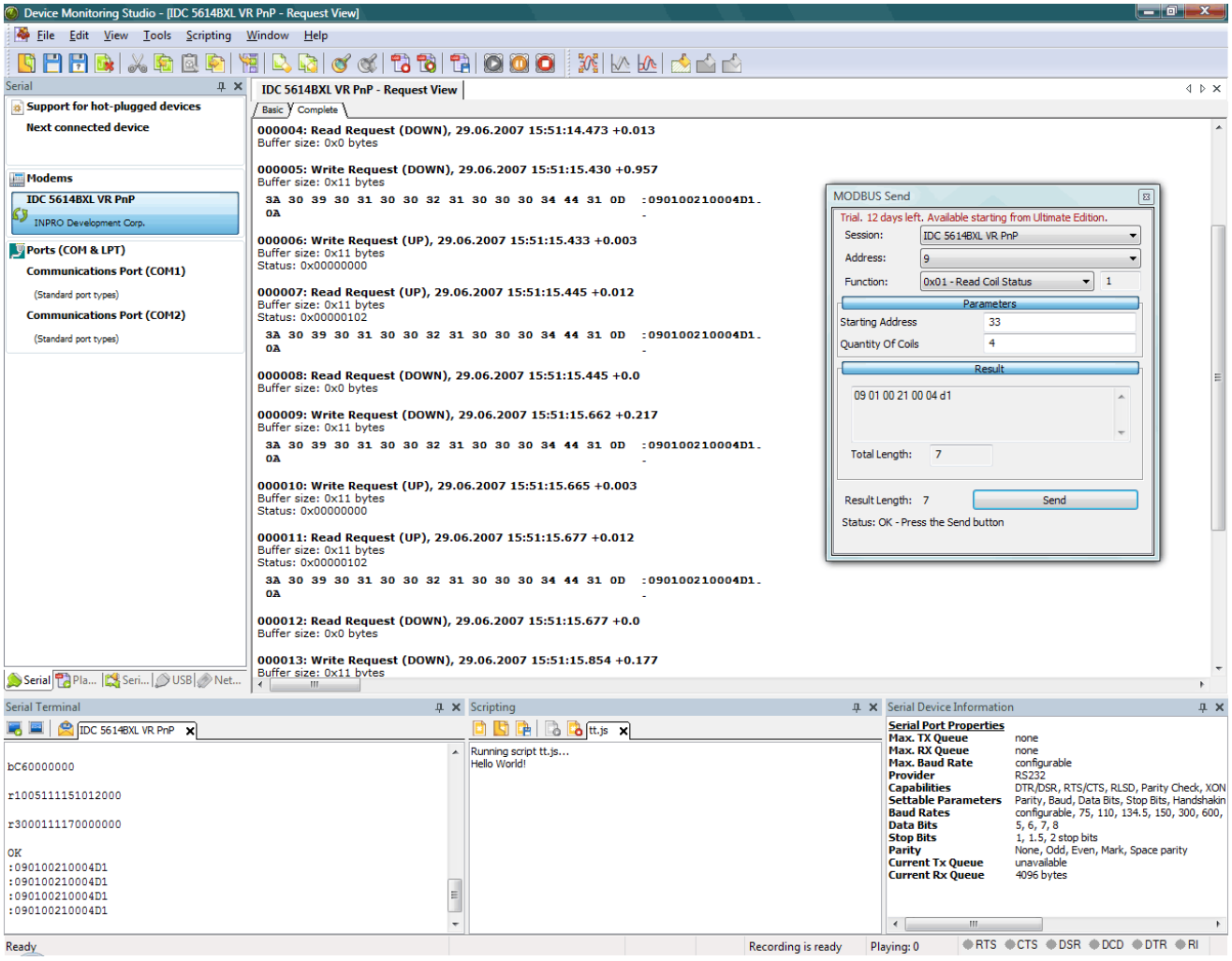
# Suggested Debugging Tools

We have found the following software to be valuable when debugging applications that involve serial communication.

## Serial Port Monitor

http://www.hhdsoftware.com/serial-monitor

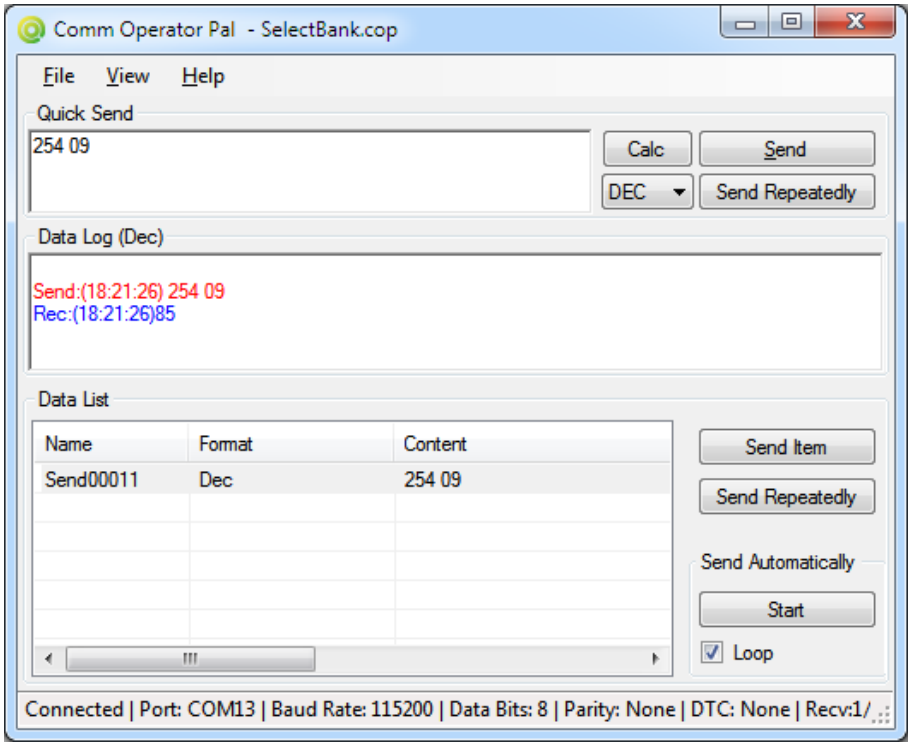
Serial Port Monitor allows you to capture, display, analyze, record and replay all serial port data exchanged between the Windows application and the serial device. It can be successfully used in application development, device driver or serial hardware development and offers the powerful platform for effective coding, testing and optimization.



## Comm Operator Pal

http://www.serialporttool.com/CommPalInfo.htm

**Comm Operator Pal** is a free tool to test and debug RS232 devices that communicated with serial port, TCP/IP or UDP. It supports data in Text, Decimal and Hex format. Data can be sent in a list automatically. Single data can be sent repeatedly.






LORD Corporation
MicroStrain® Sensing Systems
459 Hurricane Lane,
Unit 102
Williston, VT 05495 USA
www.microstrain.com
ph: 800-449-3878
fax: 802-863-4093
support@microstrain.com
sales@microstrain.com