# leJOS (/p/lejos/)

**Run Java on the Lego EV3, NXT or RCX**

**Brought to you by: bbagnall (/u/bbagnall/), gloomyandy (/u/gloomyandy/), jhsolorz (/u/jhsolorz/), lgriffiths (/u/lgriffiths/), and 2 others (/p/lejos/_members/)**



**Authors:** (/u/skoehler/)

# UART Sensor Protocol

This page is a documentation of the UART protocol spoken by LEGO EV3 sensors. It was derived from publicly available source code and reverse engineering. This is work in progress.

## General Rules

The EV3 and attached sensors send and receive messages. There are four types of messages:

- System message
- Command message
- Info message
- Data message

Each message follows one of the following structures:

- message byte (system messages)
- message byte, payload, **checksum** byte (command and data messages)
- message byte, info byte, payload, **checksum** byte (info messages)

The **checksum** byte is the XOR over the message byte, the info byte (if present), all bytes of the payload, and 0xFF.

The message byte has a special structure: 0bXXLLLYYY. The two most significant bits indicate the type of the message:

| XX | Description |
|---|---|
| 00 | System message |
| 01 | Command message |
| 10 | Info message |
| 11 | Data message |

Bits 3-5 of the message byte indicate the length of the payload. To be more precise, the size of the payload is exactly $2^{0bLLL}$ bytes. Thus, the length of a payload must always be a power of 2 and the protocol allows for payloads of at most 128 bytes while actual implementations seem to assume a maximum payload length of 32 bytes. Including the **checksum** byte, command and data messages thus have a total length of $2^{0bLLL}+2$ bytes and info messages have a total size of $2^{0bLLL}+3$ bytes. The only exception to this rule are system messages. They have a total length of 1 byte, while bits 3-5 of the message byte are zero (i.e., LLL=000).

For command and system messages, the least significant 3 bits (i.e., YYY) indicate the subtype of the message (see the tables below). For info and data messages, 0bYYY is a mode number.

Communication with a sensor starts at a speed of 2400 baud. The baud rate is changed to a different speed after the initial handshake. The final connection speed is specified by the sensor.

Integers larger than 8 bit are encoded using little endian.



## System Messages

| Message Byte | Description |
|---|---|

| Message Byte | Description |
|---|---|
| 0b00000000 | SYNC |
| 0b00000010 | NACK |
| 0b00000100 | ACK |
| 0b00LLL110 | ESC (reserved for future use) |

The meaning of bits 3-5 of the message byte of the ESC system message is **unclear**.

The purpose of the SYNC system message is **unclear**. It seems to be ignored by most implementations.

## Command Messages

| Message Byte | Payload | Description |
|---|---|---|
| 0b01000000 | T | TYPE: sensor type<br>T is the type of the sensor (0-255) |
| 0b01001001 | M, V | MODES: sensor modes<br>M+1 is the number of modes supported (1-8)<br>V+1 is the number of modes to be shown (1-M) |
| 0b01010010 | SSSS | SPEED: maximum sensor baud rate<br>SSSS (32 bit integer) is the maximum baud rate supported by the sensor |
| 0b01000011 | M | SELECT: change sensor mode<br>M specifies the desired sensor mode (0-7) |
| 0b01LLL100 | <data> | WRITE: Send data to the sensor<br><data> consists of exactly $2^{0bLLL}$ bytes |

The purpose of the WRITE command message is **unclear**. The format of the payload may be sensor specific.

## Info Messages

| Message Byte | Info Byte | Payload | Description |
|---|---|---|---|
| 0b10LLLMMM | 0 | <string> | NAME: name of mode 0bMMM<br><string> is an ASCII string, padded with zeros to a length of exactly $2^{0bLLL}$ characters |
| 0b10011MMM | 1 | LLLL, HHHH | RAW: range of raw sensor readings<br>LLLL (32bit float) is the lowest raw value the sensor returns<br>HHHH (32 bit float) is the highest raw value the sensor returns |
| 0b10011MMM | 2 | LLLL, HHHH | PCT: range of readings in percent<br>LLLL (32bit float) percent value corresponding to the lowerst raw value<br>HHHH (32 bit float) percent value corresponding to the hightest raw value |
| 0b10011MMM | 3 | LLLL, HHHH | SI: range of readings in SI units<br>LLLL (32bit float) value in SI units corresponding to the lowerst raw value<br>HHHH (32 bit float) value in SI units corresponding to the hightest raw value |
| 0b10LLLMMM | 4 | <string> | SYMBOL: name of the SI unit<br><string> is an ASCII string, padded with zeros to a length of exactly $2^{0bLLL}$ characters |
| 0b10010MMM | 0x80 | S, T, F, D | FORMAT: format of the sensor data in mode 0bMMM<br>S: the number of items (at least 1)<br>T: the data type of the items (see table)<br>F: the number of digits to show (0-15, including decimals and the decimal point)<br>D: the number of decimals to show (0-15) |



As the payload of data messages must not exceed 32 bytes, the parameter S of a FORMAT message must satisfy the following constraints:

| Data Type | Description | Maximum Value for S in FORMAT Message |
|---|---|---|
| 0 | 8 bit integer | 32 |
| 1 | 16 bit integer | 16 |
| 2 | 32 bit integer | 8 |
| 3 | 32 bit floating point | 8 |

## Data Messages

| Message Byte | Payload | Description |
|---|---|---|
| 0b11LLLMMM | <data> | DATA: sensor data<br><data> contains the raw sensor readings for mode 0bMMM, padded to a length of exactly $2^{0bLLL}$ bytes. |

TODO explain structure of <data>

## Messages from Host to Sensor

- System messages: ACK, NACK
- Command messages: SELECT, WRITE

## Message from Sensor to Host

- System messages: ACK, SYNC
- Command messages: TYPE, MODES, SPEED
- Info messages: NAME, RAW, PCT, SI, SYMBOL, FORMAT
- Data messages: DATA

## Protocol Overview

| Host | Sensor |
|---|---|
| baud rate is 2400 | baud rate is 2400 |
| | send TYPE, MODES, and SPEED messages |
| | per mode: send NAME, RAW, PCT, SI, and FORMAT messages |
| wait for ACK | send ACK |
| send ACK | wait for ACK |
| change baud rate | change baud rate |
| | DATA |
| | DATA |
| NACK | |
| | DATA |
| | DATA |
| SELECT | |
| | DATA |
| | DATA |
| NACK | |

| Host | Sensor |
|---|---|
| | DATA |
| | DATA |

In an initial phase, the sensor sends a TYPE, MODES, and SPEED message followed by several info messages for each mode. Finally, the sensor sends an ACK and waits for an ACK by the host. This concludes the initial phase.

The MODES command message is optional. A default of one mode is assumed (M=V=0). The SPEED command message is also optional. The default speed is **unclear**. The info messages for the mode with the highest number must be sent first. Per mode, the NAME message must be the first and the FORMAT message must be the last info message sent by the sensor. The info messages RAW, PCT, and SI are optional. The following defaults are used when the info message are not sent:

- RAW: 0.0 - 1023.0
- PCT: 0.0 - 100.0
- SI: 0.0 - 1.0

**Unclear:** The last FORMAT messages indicates default mode? Available material strongly suggests sending mode info in descending order (by mode number) in which case mode 0 is always default.

After the initial phase, the connection speed is switched to the one specified by the sensor. The sensor regularly sends DATA messages to the host. The host in turn regularly sends a NACK message. The host may use SELECT messages to switch the sensor to a different mode.

## Timing

There must be a delay of at least 10ms between consecutive FORMAT and NAME messages.

A sensor must send a data message at least every 100ms, but at most every 1ms.
A sensor should also send a data message after receiving a NACK.

The host should send a NACK every 100ms.

## Error Handling

A sensor should reset if it does not receiving a NACK for 1000ms. Also, if it does not receive the initial ACK within 80ms after sending the ACK to the host, then it should reset.

If the sensor does not send DATA messages for too long or too many DATA messages are invalid, then the host may stop sending any NACKs. As a result, the sensor will reset. The host may also withhold the initial ACK if any of the messages sent by the sensor in the initial phase were invalid. Again, the sensor will reset.

## Other Issues and Broken Sensors

- The protocol does not include an acknowledgement to commands like SELECT. Thus, the host must check subsequent data messages to see whether the sensor actually switched modes.
- ev3dev drivers claims that IR sensor (type 33) sends checksum byte after SYNC message (violates protocol).
- When in RGB mode (mode 4), the color sensor (type 29) is known to send DATA messages with an invalid checksum byte.