LEGO Powered Up UART Protocol

The LEGO Powered Up UART sensor protocol seems to be based on the EV3 UART sensor protocol with some new extensions. Information on this page can be assumed to apply to both EV3 UART sensors and Powered Up UART devices unless it is explicitly stated otherwise.

Note: Identifier names used here come from lms2012.h and d_uart_mod.c. Other names are inspired by LWP3. Some new identifier names are invented. Identifiers currently used by Pybricks are available from lump.h.

Conventions: In the format specifiers below, `[]` means that the enclosed content is optional and `<>` means that the content is variable and should be replaced with a suitable value.

UART Device Synchronization

When a sensor is powered on, it will start sending data at 2400 baud. The data being contains information about the device and its modes of operation. This sequence is sent repeatedly until the device receives an ACK from the programmable brick it is connected to.

The programmable brick watches for `CMD_TYPE`, which is always the first message in the sequence of data. Once this message has been identified, then the brick can decode the remaining messages in the sequence.

The sequence starts with `MESSAGE_CMD` messages that give information about the device followed by `MESSAGE_INFO` commands that give information about each mode. The last mode in the sequence is considered the default mode. In practice, the modes are enumerated from highest to lowest (0 always being the lowest).

The sequence sent by the device is terminated by `BYTE_ACK` and `BYTE_SYNC`. If the there were no errors reading any of the messages,the brick will reply with `BYTE_ACK`. Both devices will then change baud rates (the actual rate is contained in one of the messages) and the device will start sending sensor data to the brick. The brick sends `BYTE_NACK` every 100 milliseconds to let the device know it is still there. If the device does not receive `BYTE_NACK`, it will reset and go back to sending the informational messages at 2400 baud.

Newer LPF2 I/O devices can send information messages at 115200 baud instead of 2400 baud. A `CMD_SPEED` message with the baud rate (115200) must be sent to the I/O device before it starts transmitting information messages. If the I/O device supports this feature, it will reply with a `BYTE_ACK` message. If no `BYTE_ACK` is received, then it is assumed the feature is not supported and it is expected that the information messages will be received at 2400 baud.

Message Format

The general message format is `HEADER [DATA [...]] CHECKSUM`. That is, a one-byte header, followed by 0 to 32 data bytes and concluding with a checksum.

Bits 7-6 of the header are the message type. Bits 5-3 of the header are the message size and bits 2-0 are the command or mode depending on the message type.

The checksum is computed by XORing `0xff` with each byte of the message.

System Messages

System messages (`MESSAGE_SYS`) are a single byte and don't include a checksum. As mentioned above `BYTE_ACK` and `BYTE_SYNC` are sent at the end of the device information sequence. `BYTE_NACK` is sent to the device after synchronization at a regular interval (100 milliseconds) to prevent the device from resetting.

Note: The LEGO EV3 Infrared sensor appears to send a checksum (`0xff`) after `BYTE_SYNC`.

Command Messages

Command messages (`MESSAGE_CMD`) part of the device information sequence received from the device with the exception of `CMD_SELECT` and `CMD_WRITE` which are used to control the device (sent from brick to device).

CMD_TYPE

`CMD_TYPE` tells the brick what type of device this is. This command is required and must be the first command in the sequence sent by a device.

The message is formatted as follows::

```
MESSAGE_CMD | LENGTH_1 | CMD_TYPE, <type-id>, <checksum>
```

The values for (`<type-id>`) can be found in the Assigned Numbers document.

Example, LEGO BOOST Color and Distance sensor:

```
0x40, 0x25, 0x9a
  ^     ^     ^
  |     |     checksum
  |     type ID 37
  MESSAGE_CMD | LENGTH_1 | CMD_TYPE
```

CMD_MODES

`CMD_MODES` tells the brick how many modes this device has. This message is required.

The message is formatted as follows::

```
MESSAGE_CMD | LENGTH_<n> | CMD_MODES, <modes>, [<views>, [<modes2>,
<views2>,]] <checksum>
```

`<modes>` is the total number of modes minus one (limited to a max value of 7). `<views>` is the number of modes (also minus one) that can be used in Port View or Data Logger (limited to a max value of 7). If `<views>` is not provided, it is implied that it has the same value as `<modes>` . `<modes2>` and `<views2 >` are only provided by Powered Up devices. These are the same values, but are limited to a max value of 15 instead of 7). `<n>` is 1, 2 or 4 depending on how many values are provided.

Example, LEGO BOOST Color and Distance sensor:

```
0x51, 0x07, 0x07, 0x0a, 0x07, 0xa3
  ^     ^     ^     ^     ^     ^
  |     |     |     |     |     checksum
  |     |     |     |     8 views
  |     |     |     11 modes
  |     |     8 views (for EV3)
  |     8 modes (for EV3)
  MESSAGE_CMD | LENGTH_4 | CMD_MODES
```

Example, LEGO EV3 Color sensor:

```
0x49, 0x05, 0x02, 0xb1
  ^     ^     ^     ^
  |     |     |     checksum
  |     |     3 views
  |     6 modes
  MESSAGE_CMD | LENGTH_2 | CMD_MODES
```

CMD_SPEED

`CMD_SPEED` tells the brick the baud rate to use after synchronizing. This message is optional. If omitted, it is implied that the device will remain at 2400 baud. In practice, this message is always provided.

The message is formatted as follows::

```
MESSAGE_CMD | LENGTH_4 | CMD_SPEED, <speed>, <checksum>
```

`<speed>` is a 32-bit little-endian integer value.

Example, LEGO BOOST Color and Distance sensor:

```
0x52, 0x00, 0xc2, 0x01, 0x00, 0x6e
  ^     ^_________________^     ^
  |     |                       checksum
  |     |
  |     speed = 115200
  MESSAGE_CMD | LENGTH_4 | CMD_SPEED
```

CMD_SELECT

`CMD_SELECT` is used to set the mode of the device.

The message is formatted as follows::

```
MESSAGE_CMD | LENGTH_1 | CMD_SELECT, <mode>, <checksum>
```

`<mode>` is the mode.

Example, LEGO BOOST Color and Distance sensor:

```
0x43, 0x02, 0xbe
  ^     ^     ^
  |     |     checksum
  |     mode = 2 ("COUNT")
  MESSAGE_CMD | LENGTH_1 | CMD_SELECT
```

CMD_WRITE

`CMD_WRITE` writes arbitrary data to the sensor.

The message is formatted as follows::

```
MESSAGE_CMD | LENGTH_<n> | CMD_WRITE, <data>, <checksum>
```

`<data>` is `<n>` bytes of device-specific data.

Example, LEGO EV3 Gyro sensor:

```
0x44, 0x17, 0xac
  ^     ^     ^
  |     |     checksum
  |     reset command
  MESSAGE_CMD | LENGTH_1 | CMD_WRITE
```

Note: the value for the reset command comes from EV3-G software. Not sure if it actually does anything.

CMD_VERSION

`CMD_VERSION` receives the firmware version and hardware version of the device. This message is optional. It is not present on EV3 sensors.

The message is formatted as follows::

```
MESSAGE_CMD | LENGTH_8 | CMD_VERSION, <fw-version>, <hw-version>,
<checksum>
```

`<fw-version>` and `<hw-version>` are 32-bit little-endian BCD values representing the firmware and hardware versions of a device, respectively. Each The MSB contains the major and minor revision values. The next byte contains the bug fix revision and the two LSBs contain the build number.

Example, LEGO BOOST Color and Distance sensor:

```
0x5f, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x10, 0xa0
  ^     ^_________________^     ^_________________^     ^
  |     |                       |                       checksum
  |     |                       hw-version = 1.0.00.0000
  |     fw-version = 1.0.00.0000
  MESSAGE_CMD | LENGTH_8 | CMD_VERSION
```

Mode Information Messages

Mode information messages (`MESSAGE_INFO`) contain information about a specific device mode.

INFO_NAME

`INFO_NAME` gives the name of the mode. This is limited to 11 characters (not including null termination). This message is required.

This message is formatted as follows:

```
MESSAGE_INFO | LENGTH_<n> | MODE_<m>, INFO_NAME [| INFO_MODE_PLUS_8],
<name>, <checksum>
```

`<name>` is the device name encoded using ASCII characters. The length `<n>` must be a power of 2, so unused data bytes should be set to `0x00` . The data does not need to include a null terminator, e.g. if the name is 4 characters, the length can be set to 4.

The mode `<m>` is limited to 7. If the mode is greater than 7, `INFO_MODE_PLUS_8` must be set in the second byte of the message and `<m>` must be set to the mode minus 8.

Example, LEGO BOOST Color and Distance sensor:

```
0x9a, 0x00, 0x43, 0x4f, 0x55, 0x4e, 0x54, 0x00, 0x00, 0x00, 0x6d
  ^     ^     ^_____________________________________^     ^
  |     |     |                                           checksum
  |     |     "COUNT\0\0\0"
  |     INFO_NAME
  MESSAGE_INFO | LENGTH_8 | MODE_2
```

Example, LEGO BOOST Color and Distance sensor with mode > 7:

```
0x98, 0x20, 0x53, 0x50, 0x45, 0x43, 0x20, 0x31, 0x00, 0x00, 0x53
  ^     ^     ^_____________________________________^     ^
  |     |     |                                           checksum
  |     |     "SPEC 1\0\0"
  |     INFO_NAME | INFO_MODE_PLUS_8
  MESSAGE_INFO | LENGTH_8 | MODE_0
```

Newer I/O devices may also supply 6 bytes of information after the name. In this case, the name will be 5 bytes or less, followed by a zero terminator and the last 6 bytes will be the motor info.

Example, LEGO Technic Large Linear Motor:

```
0xA0, 0x00, 0x50, 0x4F, 0x57, 0x45, 0x52, 0x00, 0x30, 0x00, 0x00,
0x00, 0x05, 0x04, 0x00, 0x00, 0x00, 0x00, 0x31
  ^     ^     ^_____________________________^
^_____________________________^     ^_________________^     ^
  |     |     "POWER\0"                           flags
padding                checksum
  |     INFO_NAME
  MESSAGE_INFO | LENGTH_16 | MODE_0
```

Flags are as follows (? means bit has been seen but function is unknown):

Byte	Bit	Description
0	7	requires constant power on pin 2
0	6	requires constant power on pin 1
0	5	is a motor
0	4	POWER
0	3	
0	2	POS

Byte	Bit	Description
0	1	APOS
0	0	SPEED
1	7	
1	6	CALIB
1	5	
1	4	
1	3	
1	2	
1	1	
1	0	
2	-	
3	-	
4	7	
4	6	
4	5	
4	4	
4	3	
4	2	?
4	1	
4	0	uses power on pins 1 and 2
5	7	?
5	6	
5	5	
5	4	
5	3	
5	2	?
5	1	?

Byte	Bit	Description
5	0	

INFO_RAW

`INFO_RAW` provides information about the scaling of the raw sensor value for a mode. This information should be disregarded since UART devices always send scaled data values. This message is optional. If omitted, the min and max values are assumed to be 0.0 and 1023.0.

This message is formatted as follows:

```
MESSAGE_INFO | LENGTH_8 | MODE_<m>, INFO_RAW [| INFO_MODE_PLUS_8],
<min>, <max>, <checksum>
```

`<min>` and `<max>` are 32-bit little-endian IEEE 754 floating point values.

The mode `<m>` is limited to 7. If the mode is greater than 7, `INFO_MODE_PLUS_8` must be set in the second byte of the message and `<m>` must be set to the mode minus 8.

Example, LEGO BOOST Color and Distance sensor:

```
0x9a, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc8, 0x42, 0xee
  ^     ^     ^___________________^    ^___________________^      ^
  |     |     |                        |                        checksum
  |     |     |                       max = 100.0
  |     |     min = 0.0
  |     INFO_RAW
  MESSAGE_INFO | LENGTH_8 | MODE_2
```

INFO_PCT

`INFO_PCT` provides information on scaling the sensor value for a mode to a percentage. Basically, this is either going to be 0 to 100% or -100% to 100%. This message is optional. If omitted, the values are assumed to be 0.0 and 100.0.

This message is formatted as follows:

```
MESSAGE_INFO | LENGTH_8 | MODE_<m>, INFO_PCT [| INFO_MODE_PLUS_8],
<min>, <max>, <checksum>
```

`<min>` and `<max>` are 32-bit little-endian IEEE 754 floating point values.

The mode `<m>` is limited to 7. If the mode is greater than 7, `INFO_MODE_PLUS_8` must be set in the second byte of the message and `<m>` must be set to the mode minus 8.

Example, LEGO BOOST Color and Distance sensor:

```
0x9a, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc8, 0x42, 0xed
  ^     ^     ^___________________^    ^___________________^      ^
  |     |     |                        |                        checksum
  |     |     |                        max = 100.0
  |     |     min = 0.0
  |     INFO_PCT
  MESSAGE_INFO | LENGTH_8 | MODE_2
```

INFO_SI

`INFO_SI` provides information on the scaled data value for a mode. For some sensors, this may be the min and max possible values. For other sensors without such bounds, the max value may indicate, for example, one rotation of a motor (360.0). This message is optional. If omitted, the values are assumed to be 0.0 and 1023.0.

This message is formatted as follows:

```
MESSAGE_INFO | LENGTH_8 | MODE_<m>, INFO_SI [| INFO_MODE_PLUS_8],
<min>, <max>, <checksum>
```

`<min>` and `<max>` are 32-bit little-endian IEEE 754 floating point values.

The mode `<m>` is limited to 7. If the mode is greater than 7, `INFO_MODE_PLUS_8` must be set in the second byte of the message and `<m>` must be set to the mode minus 8.

Example, LEGO BOOST Color and Distance sensor:

```
0x9a, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc8, 0x42, 0xec
  ^     ^     ^___________________^    ^___________________^      ^
  |     |     |                        |                        checksum
  |     |     |                        max = 100.0
  |     |     min = 0.0
  |     INFO_SI
  MESSAGE_INFO | LENGTH_8 | MODE_2
```

INFO_UNITS

`INFO_UNITS` provide a string giving the units of measurement for a mode. The value is limited to 4 characters (not including the null terminator). This message is optional. If omitted, it is assumed to be an empty string.

This message is formatted as follows:

```
MESSAGE_INFO | LENGTH_<n> | MODE_<m>, INFO_UNITS [| INFO_MODE_PLUS_8],
<unit>, <checksum>
```

`<unit>` is the units of measurement encoded using ASCII characters. The length `<n>` must be a power of 2, so unused data bytes should be set to `0x00`. The data does not need to include a null terminator, e.g. if the name is 4 characters, the length can be set to 4.

The mode `<m>` is limited to 7. If the mode is greater than 7, `INFO_MODE_PLUS_8` must be set in the second byte of the message and `<m>` must be set to the mode minus 8.

Example, LEGO BOOST Color and Distance sensor:

```
0x92, 0x04, 0x43, 0x4e, 0x54, 0x00, 0x30
  ^     ^     ^__________________^    ^
  |     |     |                       checksum
  |     |     unit = "CNT\0"
  |     INFO_UNITS
  MESSAGE_INFO | LENGTH_4 | MODE_2
```

Note: The EV3 sources header files use the name `INFO_SYMBOL` instead of `INFO_UNITS` for this feature.

INFO_MAPPING

`INFO_MAPPING` provides mode mapping information for Powered Up devices. This message is omitted on EV3 sensors.

The message is formatted as follows::

```
MESSAGE_INFO | LENGTH_2 | MODE_<m>, INFO_MAPPING [| INFO_MODE_PLUS_8],
<input>, <output>, <checksum>
```

`<input>` and `<output>` are 8-bit flags.

Bit	Description
7	Supports NULL value
6	Supports Functional Mapping 2.0+
5	N/A

Bit	Description
4	ABS (Absolute [min..max])
3	REL (Relative [-1..1])
2	DIS (Discrete [0, 1, 2, 3])
1	N/A
0	N/A

The mode `<m>` is limited to 7. If the mode is greater than 7, `INFO_MODE_PLUS_8` must be set in the second byte of the message and `<m>` must be set to the mode minus 8.

Example, LEGO BOOST Color and Distance sensor:

```
0x8a, 0x05, 0x08, 0x00, 0x78
  ^     ^     ^     ^     ^
  |     |     |     |     checksum
  |     |     |     output = 0
  |     |     input = 8
  |     INFO_MAPPING
  MESSAGE_INFO | LENGTH_2 | MODE_2
```

INFO_MODE_COMBOS

`INFO_MODE_COMBOS` provides mode combination information for Powered Up devices. This message appears after the `INFO_FORMAT` message on mode 0 only, making it more akin to `CMD_...` instead of `INFO_...`. This message is omitted on EV3 sensors.

The data values 16-bit flags with each bit representing a mode. The number of data values will depend on the device. If the number of data values does not fit the length (since length can only be power of 2), then zero values will be added to pad the remaining length. There is a limit of 8 mode combination values since messages are limited to 32 bytes.

The message is formatted as follows:

```
MESSAGE_INFO | LENGTH_<l> | MODE_8, INFO_MODE_COMBOS, <data0>,
[<data1>, [...] ], <checksum>
```

`<data1>` and `<data2>` are unknown 8-bit values.

Example, LEGO BOOST Color and Distance sensor:

```
0x88, 0x06, 0x4f, 0x00, 0x3e
  ^     ^     ^_____^     ^
  |     |     |           checksum
  |     |     combos[0] = 0x004f (modes 0, 1, 2, 3 and 6)
  |     INFO_MODE_COMBOS
  MESSAGE_INFO | LENGTH_2 | MODE_0
```

INFO_FORMAT

`INFO_FORMAT` provide information about the data format and presentation for a mode. This mode is required and must be the last `INFO_...` message for a mode.

This message is formatted as follows:

```
MESSAGE_INFO | LENGTH_4 | MODE_<m>, INFO_FORMAT [| INFO_MODE_PLUS_8],
<data-sets>, <format>, <figures>, <decimals>, <checksum>
```

`<data-sets>` is the number of sensor data values for this mode. `<format>` is the data format (one of `DATAx`). `<data-sets>` times the size of `<format>` must not exceed 32-byte. For example 32 8-bit values are allowed, but only 8 32-bit values are allowed. `<figures>` is the number of characters, including the decimal point, to be displayed. `<decimals>` is the number of digits after the decimal point. When `<format>` is an integer type and `<decimals>` is non-zero, this implies a fixed point number. For example, if `<decimals>` is 1 and `<format>` is `DATA16` , then the actual sensor data value is the integer value divided by 10.

The mode `<m>` is limited to 7. If the mode is greater than 7, `INFO_MODE_PLUS_8` must be set in the second byte of the message and `<m>` must be set to the mode minus 8.

Example, LEGO BOOST Color and Distance sensor:

```
0x92, 0x80, 0x01, 0x02, 0x04, 0x00, 0x30
  ^     ^     ^     ^     ^     ^     ^
  |     |     |     |     |     |     checksum
  |     |     |     |     |     decimals = 0
  |     |     |     |     figures = 4
  |     |     |     format = DATA32
  |     |     data-sets = 1
  |     INFO_FORMAT
  MESSAGE_INFO | LENGTH_4 | MODE_2
```

Data Messages

Data messages (`MESSAGE_DATA`) are sent from the device to the brick after the synchronization process has been complete (when brick sends ACK to device after parsing info messages). Messages are sent whenever the device has new sensor info or in response to a NACK (keep-alive) message from the brick.

LPF2 devices also have the ability to write data to the device for modes that support it, for example, controlling the color of the light on a color sensor.

MESSAGE_DATA

This message is formatted as follows:

```
MESSAGE_DATA | LENGTH_<n> | MODE_<m>, <data>, <checksum>
```

The size and format of `<data>` determined by the `<data-sets>` and `<format>` values returned in the `INFO_FORMAT` message. `<n>` is the length of `<data>` in bytes. Since `<n>` must be a power of 2, there may be some extra bytes in `<data>` that should be ignored.

Example, LEGO BOOST Color and Distance sensor:

```
0xC0, 0x00, 0x3f
 |     |     |
 |     |     checksum
 |     color index = 0
 MESSAGE_DATA | LENGTH_1 | MODE_0
```

Writing Data

LPF2 sensors have the added capability of being able to write formatted data to a specific mode of the sensor. The format is the same as the one for data received. The messages are actually two combined message to account for the extended mode on LPF2 devices. The first message has the format:

```
MESSAGE_CMD | LENGTH_1 | CMD_EXT_MODE, EXT_MODE_<m>, <checksum>
```

`<m>` is either 0 or 8 and will be added to the mode in the `MESSAGE_DATA` message to get the actual mode.

The second message has the same format as data messages that are received from the device (see previous section for details):

```
MESSAGE_DATA | LENGTH_<n> | MODE_<m>, <data>, <checksum>
```

Example, LEGO BOOST Color and Distance sensor:

```
0x46, 0x00, 0xb9, 0xC5, 0x00, 0x3a
 ^     ^     ^     ^     ^     ^
 |     |     |     |     |     checksum
 |     |     |     |     color index = 0
 |     |     |     MESSAGE_DATA | LENGTH_1 | MODE_5
 |     |     checksum
 |     EXT_MODE_0
 MESSAGE_CMD | LENGTH_1 | CMD_EXT_MODE
```

Definitions

```
// FIRST BYTE

// bits 7-6
#define   MESSAGE_SYS                   0x00    // System message
0b00 << 6
#define   MESSAGE_CMD                   0x40    // Command message
0b01 << 6
#define   MESSAGE_INFO                  0x80    // Info message
0b10 << 6
#define   MESSAGE_DATA                  0xC0    // Data message
0b11 << 6

// bits 5-3
#define   LENGTH_1                      0x00    // 1 byte
0b000 << 3
#define   LENGTH_2                      0x08    // 2 bytes
0b001 << 3
#define   LENGTH_4                      0x10    // 4 bytes
0b010 << 3
#define   LENGTH_8                      0x18    // 8 bytes
0b011 << 3
#define   LENGTH_16                     0x20    // 16 bytes
0b100 << 3
#define   LENGTH_32                     0x28    // 32 bytes
0b101 << 3

// MESSAGE_SYS bits 2-0
#define   BYTE_SYNC                     0x00    // Synchronization
byte
#define   BYTE_NACK                     0x02    // Not acknowledge
byte (keep alive)
#define   BYTE_ACK                      0x04    // Acknowledge byte

// MESSAGE_CMD bits 2-0
#define   CMD_TYPE                      0x00    // CMD command - TYPE
(device type for VM reference)
#define   CMD_MODES                     0x01    // CMD command - MODES
(number of supported modes minus one)
#define   CMD_SPEED                     0x02    // CMD command - SPEED
(maximum communication speed)
#define   CMD_SELECT                    0x03    // CMD command -
SELECT   (select mode)
#define   CMD_WRITE                     0x04    // CMD command - WRITE
(write to device)
#define   CMD_EXT_MODE                  0x06    // CMD command -
EXT_MODE (value will be added to mode in CMD_WRITE_DATA - LPF2 only)
#define   CMD_VERSION                   0x07    // CMD command -
VERSION  (device firmware and hardware versions)

// MESSAGE_INFO and MESSAGE_DATA bits 2-0
#define   MODE_0                        0x00    // MODE 0 (or 8 if
```

```
INFO_MODE_PLUS_8 bit is set)
#define   MODE_1                        0x01    // MODE 1 (or 9 if
INFO_MODE_PLUS_8 bit is set)
#define   MODE_2                        0x02    // MODE 2 (or 10 if
INFO_MODE_PLUS_8 bit is set)
#define   MODE_3                        0x03    // MODE 3 (or 11 if
INFO_MODE_PLUS_8 bit is set)
#define   MODE_4                        0x04    // MODE 4 (or 12 if
INFO_MODE_PLUS_8 bit is set)
#define   MODE_5                        0x05    // MODE 5 (or 13 if
INFO_MODE_PLUS_8 bit is set)
#define   MODE_6                        0x06    // MODE 6 (or 14 if
INFO_MODE_PLUS_8 bit is set)
#define   MODE_7                        0x07    // MODE 7 (or 15 if
INFO_MODE_PLUS_8 bit is set)

// CMD_EXT_MODE payload
#define   EXT_MODE_0                    0x00    // mode is < 8
#define   EXT_MODE_8                    0x08    // mode is >= 8

// SECOND INFO BYTE

#define   INFO_NAME                     0x00    // INFO command - NAME
(device name)
#define   INFO_RAW                      0x01    // INFO command - RAW
(device RAW value span)
#define   INFO_PCT                      0x02    // INFO command - PCT
(device PCT value span)
#define   INFO_SI                       0x03    // INFO command - SI
(device SI  value span)
#define   INFO_UNITS                    0x04    // INFO command -
UNITS   (device SI  unit symbol)
#define   INFO_MAPPING                  0x05    // INFO command -
MAPPING (input/output value type flags)
#define   INFO_MODE_COMBOS              0x06    // INFO command -
COMBOS  (mode combinations - LPF2-only)
#define   INFO_UNK7                     0x07    // INFO command -
unknown (LPF2-only)
#define   INFO_UNK8                     0x08    // INFO command -
unknown (LPF2-only)
#define   INFO_UNK9                     0x09    // INFO command -
unknown (LPF2-only)
#define   INFO_UNK10                    0x0a    // INFO command -
unknown (LPF2-only)
#define   INFO_UNK11                    0x0b    // INFO command -
unknown (LPF2-only)
#define   INFO_UNK12                    0x0c    // INFO command -
unknown (LPF2-only)
#define   INFO_MODE_PLUS_8              0x20    // Bit flag used in
powered up devices to indicate that the mode is 8 + the mode specified
in the first byte
#define   INFO_FORMAT                   0x80    // INFO command -
```

```
FORMAT  (device data sets and format)

// INFO_FORMAT formats
#define   DATA8                         0x00    // 8-bit signed
integer
#define   DATA16                        0x01    // 16-bit little-
endian signed integer
#define   DATA32                        0x02    // 32-bit little-
endian signed integer
#define   DATAF                         0x03    // 32-bit little-
endian IEEE 754 floating point
```