



## **ANEXO 3**

### **COMANDOS DIRECTOS LEGO NXT**



# LEGO® MINDSTORMS® NXT Direct Commands

# TABLE OF CONTENTS

<b>TABLE OF CONTENTS .....</b>	<b>2</b>
<b>INTRODUCTION .....</b>	<b>3</b>
<b>OVERALL ARCHITECTURE .....</b>	<b>4</b>
Maximum Command Length .....	4
Bluetooth® messages .....	4
Optional Responses .....	4
<b>LIST OF COMMANDS .....</b>	<b>5</b>
StartProgram .....	5
StopProgram .....	5
PlaySoundFile .....	5
PlayTone .....	6
SetOutputState .....	6
SetInputMode .....	7
GetOutputState .....	8
GetInputValues .....	8
ResetInputScaledValue .....	8
MessageWrite .....	9
ResetMotorPosition .....	9
GetBatteryLevel .....	9
StopSoundPlayback .....	9
KeepAlive .....	10
LSGetStatus .....	10
LSWrite .....	10
LSRead .....	10
GetCurrentProgramName .....	11
MessageRead .....	11
<b>ERROR MESSAGE BACK TO THE HOST .....</b>	<b>12</b>

## INTRODUCTION

This document describes a sub-protocol of the main LEGO® MINDSTORMS® NXT Communication Protocol specifically designed for direct commands which make it possible to control the NXT brick from outside devices. These outside devices may be other NXT bricks, a PC, or any other Bluetooth® capable device which used the serial port profile

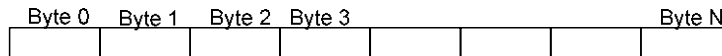
The main intent behind including this protocol is to provide a simple interface for these outside devices to utilize basic brick functionality (i.e. motor control, sensor readings, and power management) without the need to write or run specialized remote control programs on the brick. The protocol also includes an interface to send arbitrary messages to a target brick, which may be picked up and used in a user-defined NXT program

## OVERALL ARCHITECTURE

It will be possible to control the brick either through the USB communication channel or through the Bluetooth® communication channel, which both uses the LEGO® MINDSTORMS® NXT Communication protocol

For further details and explanation on the USB and Bluetooth® communication channels, please refer to LEGO® MINDSTORMS® NXT Communication protocol document

The figure below shows the general telegram architecture:



**Figure 1: General protocol architecture**

Byte 0: Telegram type, as dictated by main LEGO® MINDSTORMS® NXT Communication protocol specification. Note that for the purposes of this document, only "direct command telegrams" and "reply telegrams" are important – "system commands" are specified and handled at the main LEGO® MINDSTORMS® NXT Communication protocol document

- 0x00: Direct command telegram, response required
- 0x01: System command telegram, response required
- 0x02: Reply telegram
- 0x80: Direct command telegram, no response
- 0x81: System command telegram, no response

Byte 1 – N: the command itself or a reply, depending on telegram type

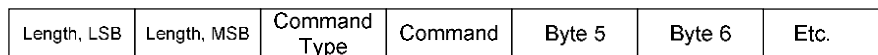
### MAXIMUMCOMMAND LENGTH

Currently, total direct command telegram size is limited to 64 bytes, including the telegram type byte as listed above. As specified in the LEGO® MINDSTORMS® NXT Communication protocol document, Bluetooth® packets have an additional two bytes for size tacked onto the front, but these are not included in this limit

### BLUETOOTH® MESSAGES

As explained above all Bluetooth® messages needs to have two bytes in front of the messages itself which indicates how many bytes the message includes. The length of the packages is counted without the two length bytes

The figure below shows a Bluetooth® message:



**Figure 2: Bluetooth® protocol packages**

### OPTIONAL RESPONSES

The LEGO® MINDSTORMS® NXT Communication protocol specification states that any incoming protocol telegram may be marked with the 0x80 mask on its telegram type byte to indicate that no response is expected. Direct commands is a primary use case for this functionality, as requiring a response on all telegrams could lead up to approximately 60 ms latency. Of course, this concept doesn't hold for all commands – for example, attempting to "GetInputValues" without requiring a response would just be wasting time

## LIST OF COMMANDS

Each of the valid commands is described in detail in the following list.

### Format Notes:

- All response packages include a status byte, where 0x00 means “success” and any non-zero value indicates a specific error condition.
- All single byte values are unsigned, unless specifically stated. Internal data type is listed for all multi-byte values and all are assumed to be little-endian as in the LEGO® MINDSTORMS® NXT Communication Protocol.
- If a legal range is not specified here explicitly, it is generally documented in the relevant module documents and/or code.
- Variable length packet fields are specified as in this example: “Byte 4 – N: Message data”, where ‘N’ is the variable size of a given field plus command overhead. ‘N’ may not exceed the Maximum Command Length mentioned above, minus 1.

### STARTPROGRAM

Byte 0: 0x00 or 0x80

Byte 1: 0x00

Byte 2 - 21: File name. Format: ASCIIZ-string with maximum size [15.3 chars] + Null terminator

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x00

Byte 2: Status Byte

### STOPPROGRAM

Byte 0: 0x00 or 0x80

~~Byte 1: 0x01~~

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x01

Byte 2: Status Byte

### PLAYSOUNDFILE

Byte 0: 0x00 or 0x80

Byte 1: 0x02

Byte 2: Loop? (Boolean: TRUE: Loop sound file indefinitely, FALSE: Play file once only)

Byte 3 - 22: File name. Format: ASCIIZ-string with maximum size [15.3 chars] + Null terminator

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x02

Byte 2: Status Byte

## PLAYTONE

Byte 0: 0x00 or 0x80

Byte 1: 0x03

Byte 2 - 3: Frequency for the tone, Hz (UWORD; Range: 200 - 14000 Hz)

Byte 4 - 5: Duration of the tone, ms (UWORD; Range: ???)

Return package:

Byte 0: 0x02

Byte 1: 0x03

Byte 2: Status Byte

## SETOUTPUTSTATE

Byte 0: 0x00 or 0x80

Byte 1: 0x04

Byte 2: Output port (Range: 0 - 2; 0xFF is special value meaning 'all' for simple control purposes)

Byte 3: Power set point (Range: -100 - 100)

Byte 4: Mode byte (Bit-field)

Byte 5: Regulation mode (UBYTE; enumerated)

Byte 6: Turn Ratio (SBYTE; -100 - 100)

Byte 7: RunState (UBYTE; enumerated)

Byte 8 - 12: TachoLimit (ULONG; 0: run forever)

Return package:

Byte 0: 0x02

Byte 1: 0x04

Byte 2: Status Byte

Valid enumeration for "Mode":

~~MOTOR\_ON~~ 0x01 Turn on the specified motor

BRAKE 0x02 Use run/brake instead of run/float in PWM

~~REGULATED~~ 0x04 Turn on the regulation

Valid enumeration for "Regulation Mode":

<del>REGULATION_MODE_IDLE</del>	0x00	No regulation will be enabled
<del>REGULATION_MODE_MOTOR_SPEED</del>	0x01	Power control will be enabled on specified output
REGULATION_MODE_MOTOR_SYNC	0x02	Synchronization will be enabled (Needs enabled on two output)

Valid enumeration for "RunState":

<del>MOTOR_RUN_STATE_IDLE</del>	0x00	Output will be idle
MOTOR_RUN_STATE_RAMPUP	0x10	Output will ramp-up
MOTOR_RUN_STATE_RUNNING	0x20	Output will be running
<del>MOTOR_RUN_STATE_RAMPDOWN</del>	0x40	Output will ramp-down

## SETINPUTMODE

Byte 0: 0x00 or 0x80

Byte 1: 0x05

Byte 2: Input port (Range: 0 - 3)

Byte 3: Sensor type (enumerated)

Byte 4: Sensor mode (enumerated)

Return package:

Byte 0: 0x02

Byte 1: 0x05

Byte 2: Status Byte

Valid enumeration for "Sensor Type":

NO_SENSOR	0x00
SWITCH	0x01
TEMPERATURE	0x02
REFLECTION	0x03
ANGLE	0x04
LIGHT_ACTIVE	0x05
LIGHT_INACTIVE	0x06
SOUND_DB	0x07
SOUND_DBA	0x08
CUSTOM	0x09
LOWSPEED	0x0A
LOWSPEED 9V	0x0B
NO_OF_SENSOR_TYPES	0x0C

Valid enumeration for "Sensor Mode":

RAWMODE	0x00
BOOLEANMODE	0x20
TRANSITIONCNTMODE	0x40
PERIODCOUNTERMODE	0x60
PCTFULLSCALEMODE	0x80
CELSIUSMODE	0xA0
FAHRENHEITMODE	0xC0
ANGLESTEPMODE	0xE0
SLOPEMASK	0x1F
MODEMASK	0xE0



## GETOUTPUTSTATE

Byte 0: 0x00 or 0x80

Byte 1: 0x06

Byte 2: Output port (Range: 0 - 2)

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x06

Byte 2: Status Byte

Byte 3: Output port (Range: 0 - 2)

Byte 4: Power set point (-100 - 100)

Byte 5: Mode (bit-field)

Byte 6: Regulation mode (UBYTE; enumerated)

Byte 7: Turn Ratio (SBYTE; -100 - 100)

Byte 8: RunState (UBYTE; enumerated)

Byte 9 - 12: TachoLimit (ULONG; Current limit on a movement in progress, if any)

Byte 13 - 16: TachoCount (SLONG; Internal count. Number of counts since last reset of the motor counter)

Byte 17 - 20: BlockTachoCount (SLONG; Current position relative to last programmed movement)

Byte 21 - 24: RotationCount (SLONG; Current position relative to last reset of the rotation sensor for this motor)

Look at the entry for SetOutputState for details about the enumerations.

## GETINPUTVALUES

Byte 0: 0x00 or 0x80

Byte 1: 0x07

Byte 2: Input port (Range: 0 - 3)

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x07

Byte 2: Status Byte

Byte 3: Input port (Range: 0 - 3)

Byte 4: Valid? (Boolean; TRUE if new data value should be seen as valid data)

Byte 5: Calibrated? (Boolean; TRUE if calibration file found and used for "Calibrated Value" field below)

Byte 6: Sensor type (enumerated)

Byte 7: Sensor mode (enumerated)

Byte 8 - 9: Raw A/D value (UWORD; device dependent)

Byte 10 - 11: Normalized A/D value (UWORD; type dependent; Range: 0 - 1023)

Byte 12 - 13: Scaled value (SWORD; mode dependent)

Byte 14 - 15: Calibrated value (SWORD; Value scaled according to calibration. CURRENTLY UNUSED.)

Look at the entry SetInputMode for details about the enumerations.

## RESETINPUTSCALEDVALUE

Byte 0: 0x00 or 0x80

Byte 1: 0x08

Byte 2: Input port (Range: 0 - 3)

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x08

Byte 2: Status Byte

## MESSAGEWRITE

Byte 0: 0x00 or 0x80

Byte 1: 0x09

Byte 2: Inbox number (0 - 9)

Byte 3: Message size

Byte 4 - N: Message data, where N = Message size + 3

Message data is treated as a string; it must include null termination to be accepted. Accordingly, message size must include the null termination byte. Message size must be capped at 59 for all message packets to be legal on USBI

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x09

Byte 2: Status Byte

## RESETMOTORPOSITION

Byte 0: 0x00 or 0x80

Byte 1: 0x0A

Byte 2: Output port (Range: 0 - 2)

Byte 3: Relative? (Boolean; TRUE: position relative to last movement, FALSE: absolute position)

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x0A

Byte 2: Status Byte

## GETBATTERYLEVEL

Byte 0: 0x00 or 0x80

~~Byte 1: 0x0B~~

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x0B

Byte 2: Status Byte

Byte 3-4: Voltage in millivolts (UWORD)

## STOPSOUNDPLAYBACK

Byte 0: 0x00 or 0x80

~~Byte 1: 0x0C~~

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x0C

Byte 2: Status Byte

## KEEPALIVE

Byte 0: 0x00 or 0x80

Byte 1: 0x0D

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x0D

Byte 2: Status Byte

Byte 3-6: Current sleep time limit, milliseconds (ULONG)

## LSGETSTATUS

Byte 0: 0x00 or 0x80

Byte 1: 0x0E

Byte 2: Port (0 - 3)

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x0E

Byte 2: Status Byte

Byte 3: Bytes Ready (count of available bytes to read)

## LSWRITE

Byte 0: 0x00 or 0x80

Byte 1: 0x0F

Byte 2: Port (0 - 3)

Byte 3: Tx Data Length (bytes)

Byte 4: Rx Data Length (bytes)

Byte 5 - N: Tx Data, where  $N = \text{Tx Data Length} + 4$

For LS communication on the NXT, data lengths are limited to 16 bytes per command. Rx Data Length MUST be specified in the write command since reading from the device is done on a master-slave basis.

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x0F

Byte 2: Status Byte

## LSREAD

Byte 0: 0x00 or 0x80

Byte 1: 0x10

Byte 2: Port (0 - 3)

Return package:

~~Byte 0: 0x02~~

Byte 1: 0x10

Byte 2: Status Byte

Byte 3: Bytes Read

Byte 3 - 19: Rx Data (padded)

For LS communication on the NXT, data lengths are limited to 16 bytes per command. Furthermore, this protocol does not support variable-length return packages, so the response will always contain 16 data bytes, with invalid data bytes padded with zeroes.

## GETCURRENTPROGRAMNAME

Byte 0: 0x00 or 0x80

Byte 1: 0x11

Return package:

Byte 0: 0x02

Byte 1: 0x11

Byte 2: Status Byte

Byte 3 - 22: File name. Format: ASCIIZ-string with maximum size [15.3 chars] + Null terminator

If no program is currently running, an error will be returned and File name field will be all zeroes.

## MESSAGEREAD

Byte 0: 0x00 or 0x80

Byte 1: 0x13

Byte 2: Remote Inbox number (0 - 9)

Byte 3: Local Inbox number (0 - 9)

Byte 4: Remove? (Boolean; TRUE (non-zero) value clears message from Remote Inbox)

Return package:

Byte 0: 0x02

Byte 1: 0x13

Byte 2: Status Byte

Byte 3: Local Inbox number (0 - 9)

Byte 4: Message size

Byte 5 - 63: Message data (padded)

Message data is treated as a string; it must include null termination. Accordingly, message size includes the null termination byte. Furthermore, return packages have a fixed size, so the message data field will be padded with null bytes

Note that the remote Inbox number may specify a value of 0-19, while all other mailbox numbers should remain below 9. This is due to the master-slave relationship between connected NXT bricks. Slave devices may not initiate communication transactions with their masters, so they store outgoing messages in the upper 10 mailboxes (indices 10-19). Use the MessageRead command from the master device to retrieve these messages.

When reading remote messages from slave devices, send the following commands:

0x05, 0x00, 0x00, 0x13, 0x0A, 0x00, 0x01 => Read Mailbox 0 from slave and clear message on slave

0x05, 0x00, 0x00, 0x13, 0x0B, 0x01, 0x01 => Read Mailbox 1 from slave and clear message on slave

## ERROR MESSAGE BACK TO THE HOST:

The information below describes how we categorize error messages; it does not describe a separate or special error message. This information will be included within the return packages.

Return package:

Byte 0: 0x02

Byte 1: Command that is receiving a reply

Byte 2, Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.

The following error messages may be received from the NXT unit:

- |   |      |
|---|------|
| • Pending communication transaction in progress       | 0x20 |
| • Specified mailbox queue is empty                    | 0x40 |
| • Request failed (i.e. specified file not found)      | 0xBD |
| • Unknown command opcode                              | 0xBE |
| • Insane packet                                       | 0xBF |
| • Data contains out-of-range values                   | 0xC0 |
| • Communication bus error                             | 0xDD |
| • No free memory in communication buffer              | 0xDE |
| • Specified channel/connection is not valid           | 0xDF |
| • Specified channel/connection not configured or busy | 0xE0 |
| • No active program                                   | 0xEC |
| • Illegal size specified                              | 0xED |
| • Illegal mailbox queue ID specified                  | 0xEE |
| • Attempted to access invalid field of a structure    | 0xEF |
| • Bad input or output specified                       | 0xF0 |
| • Insufficient memory available                       | 0xFB |
| • Bad arguments                                       | 0xFF |