Version: 1.00

LEGO® MINDSTORMS® NXT Communication Protocol

TABLE OF CONTENTS

LEGO® MINDSTORMS® NXT COMMUNICATION PROTOCOL

The LEGO® MINDSTORMS® NXT product will include new communication possibilities. The new communication possibility will enable higher communication speeds and improved wireless communication which will support new play scenarios.

The LEGO MINDSTORMS NXT includes the following communication possibilities:

- Bluetooth® communication, V2.0 with EDR.
  - Supporting the Serial Port Profile (SPP)
- USB communication, V2.0

Beside the above mentioned two main communication protocols, the LEGO MINDSTORMS NXT will also include two communication interfaces which primary purpose is communication with external embedded devices.

- One 6 wired digital communication port, maximum communication speed at 1 Mbit/s (High speed port)
- Four 6 wired digital communication ports, maximum communication speed at 9600 bit/s (Low speed ports)

The low-speed communication is using the I²C communication standard. The primary usages for the high-speed port are to enable the NXT to communicate with external devices which requires high communication speeds. We are targeting using the P-Net communication protocol (www.P-net.org) which enables multipoint data communication.

OVERALL SYSTEM COMMUNICATION LAYERS

The figure below shows the main layers in the communication stack between the embedded device and the software on the PC. As shown in the figure, it is possible to communicate between the two devices in two different ways, using wireless or using a wire.

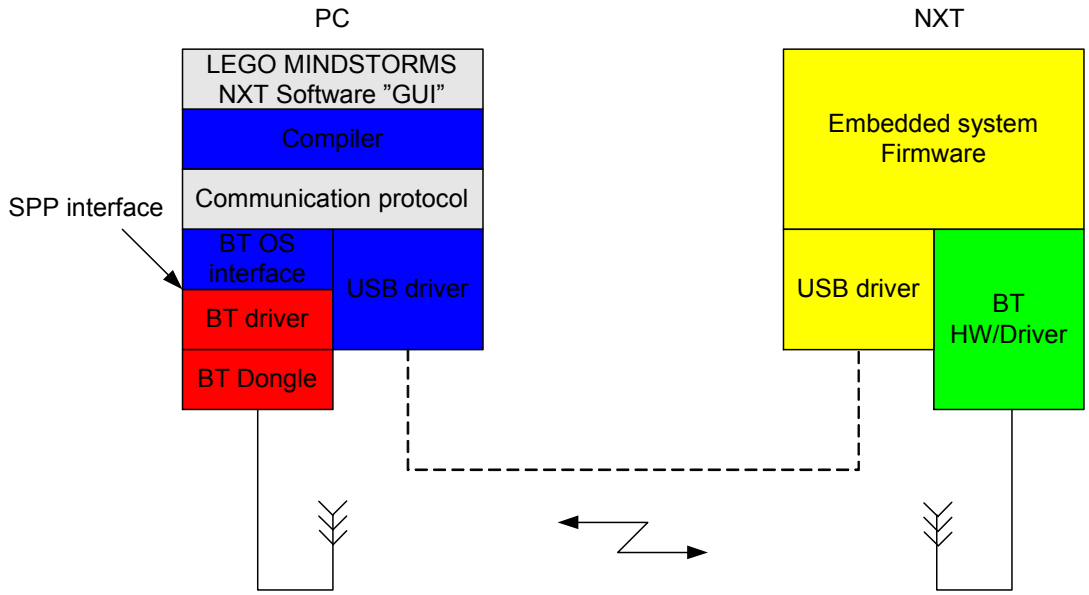

Figure 1: Communication block diagram

It will be possible to access the communication with the NXT either using the LEGO® MINDSTORMS® NXT Communication protocol, or by writing and reading raw data directly to and from the communication buffers. When writing and reading directly to and from the buffers, there is no way to check on the data being sent or read back from the NXT unit.

COMMANDS WITHIN THE COMMUNICATION PROTOCOL

This section describes the overall architecture of the LEGO® MINDSTORMS® NXT Communication protocol. This layer is placed above both the USB and the Bluetooth® communication ports because this layer will handle the data that will be the payload within the USB and the Bluetooth® communication layers.

All the calculation of checksum and setting of package numbers will be handled by the individual communication layers (i.e., either the USB communication or Bluetooth® communication layers).

The new communication protocol handles the following communication purposes:

Download files (File write):

- Download firmware *.rfw
- Download user defined programs *.rxe
- OnBrick programming (Files are generated on the brick) *.rpg
- Try-Me programs (Files are pre-programmed) *.rtm
- Sound *.rso
- Graphics *.ric

Upload files (File read):

- Upload programs *.rxe
- Upload graphics *.ric
- Upload sound *.rso
- Upload datalog files *.rdt

Delete user-defined data in the embedded system

- Delete sounds *.rso
- Delete graphics *.ric
- Delete user defined programs *.rxe
- OnBrick programming (Files are generated on the brick) *.rpg
- Try-Me programs (Files are pre-programmed) *.rtm
- Delete datalog files *.rdt

Direct communication with the NXT system:

- Get file list within NXT
- Direct commands to the virtual machine
- Messages command to mailboxes (Like the PB message in RCX 2.0)
- Data directly from the high speed port
    - Data should be written into a file on the HOST PC (This could be done using a handler which is specified to a handler. Then the communication port could be seen as a file, both as read and write).

General protocol overview, payload data from the USB or Bluetooth communication channel:

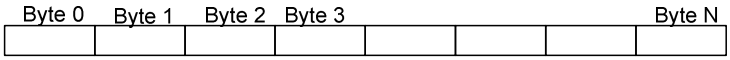

Figure 2: Protocol payload data

Byte 0: Command type. The 7th lowest bit of this byte is used for identifying the command type. Currently the system has the following command types that need to be identified. Bit 7 (MSB) is used for identifying whether the command should give a reply message or not.

- 0x00: Direct command, reply required
- 0x01: System command, reply required
- 0x02: Reply command

- 0x80: Direct command, reply not required
- 0x81: System command, reply not required

Byte 1: Command byte. Used by the loader module to identify what should happen with the data. Open, Read, Write, Delete data, Direct communication with NXT

Byte 2 - N: These bytes offer additional information.

The above figure shows the LEGO® MINDSTORMS® NXT communication protocol standard. This protocol will be wrapped into either the Bluetooth® serial profile or the USB 2.0 communication protocol.

It will be possible to download 64 bytes (USB HW buffer size) before the microcontroller needs to save the data in FLASH. This will take approximately 6 mS during which the microcontroller cannot receive additional data. Currently the system stores the received data-bytes in the buffer until it has 256 bytes of data, which correlates to one data-page within the FLASH memory of the Atmel ARM processor. When 256 bytes have been received, they are written into the FLASH memory of the Atmel ARM7 processor.

All files names in the commands below will use 19 bytes (15.3 chars). Unused characters should be filled with null terminators.

All commands will, by default, have a return package. That means that if no return package is desired, the command should explicit specify so within the command.

The commands on the following pages are used between the communicating device and the loader module within the firmware.

OPEN READ COMMAND:

Byte 0: 0x01
Byte 1: 0x80
Byte 2 - 21: The name of the file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator

Return package:
Byte 0: 0x02
Byte 1: 0x80
Byte 2, Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3, Handle: Handle number.
Byte 4: LSB of file size (Little Endian)
Byte 5:
Byte 6:
Byte 7: MSB of file size

Return packages should not be disabled for this command.

When this command returns a success, a close command is required for "closing the handle" within the brick when the handle is not needed any more. If an error is returned, the firmware will close the handle automatically.

OPEN WRITE COMMAND:

Byte 0: 0x01
Byte 1: 0x81
Byte 2 - 21: The name of the file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator
Byte 22: LSB of file size (Little Endian)
Byte 23:
Byte 24:
Byte 25: MSB of file size

Return package:
Byte 0: 0x02
Byte 1: 0x81
Byte 2, Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3, Handle: Handle number.

Return packages should not be disabled for this command.

When this command returns a success, a close command is required for "closing the handle" within the brick when the handle is not needed any more. If an error is returned, the firmware will close the handle automatically.

READ COMMAND:

Byte 0: 0x01
Byte 1: 0x82
Byte 2: Handle number
Byte 3: Number of data bytes to be read, LSB
Byte 4: Number of data bytes to be read, MSB

Return package:
Byte 0: 0x02
Byte 1: 0x82
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Handle: Handle number.
Byte 4: Number of data that have been read, LSB
Byte 5: Number of data that have been read, MSB
Byte 6 – N: Data

Return packages should not be disabled for this command.

WRITE COMMAND:

Byte 0: 0x01
Byte 1: 0x83
Byte 2: Handle number
Byte 3 – N: Data to be written into FLASH

Return package:
Byte 0: 0x02
Byte 1: 0x83
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Handle: Handle number.
Byte 4: Tells how many bytes have been written into FLASH, LSB
Byte 5: Tells how many bytes have been written into FLASH, MSB

CLOSE COMMAND:

Byte 0: 0x01
Byte 1: 0x84
Byte 2: Handle number

Return package:
Byte 0: 0x02
Byte 1: 0x84
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Handle: Handle number.

DELETE COMMAND:

Byte 0: 0x01
Byte 1: 0x85
Byte 2 - 21: The name of the file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator

Return package:
Byte 0: 0x02
Byte 1: 0x85
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3 - 22: The name of the deleted file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator

FIND FIRST:

Byte 0: 0x01
Byte 1: 0x86
Byte 2 - 21: The filename and/or extension types on file(s) to search. See Wildcard to know which wildcards are allowed. + Null terminator.

Return package:
Byte 0: 0x02
Byte 1: 0x86
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Handle: Handle Number
Byte 4 - 23: The name of the file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator
Byte 24: LSB of file size (Little Endian)
Byte 25:
Byte 26:
Byte 27: MSB of file size

Return packages should not be disabled for this command.

When no files exist within the system, an error message is returned within the return packages saying: “File not found”.

When this command returns a success, a close command is required for “closing the handle” within the brick when handle is not needed any more. If an error is returned, the firmware will close the handle automatically.

FIND NEXT:

Byte 0: 0x01
Byte 1: 0x87
Byte 2: Handle number from the previous found file or from the Find First command.

Return package:
Byte 0: 0x02
Byte 1: 0x87
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Handle: Handle number
Byte 4 - 23: Tell the name of the file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator
Byte 24: LSB of file size (Little Endian)
Byte 25:
Byte 26:
Byte 27: MSB of file size

Return packages should not be disabled for this command.

When no files exits within the system, an error message is returned with the return packages saying: “File not found”.

When this command returns a success, a close command is required for “closing the handle” within the brick when handle is not needed any more. If an error is returned, the firmware will close the handle automatically.

GET FIRMWARE VERSION:

Byte 0: 0x01
Byte 1: 0x88

Return package:
Byte 0: 0x02
Byte 1: 0x88
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Minor version of protocol
Byte 4: Major version of protocol
Byte 5: Minor version of firmware
Byte 6: Major version of firmware

Return packages should not be disabled for this command.

This will allow the possibility of having different versions of the firmware.

For example, if the bytes are Byte 3: 0x02, Byte 4: 0x01, Byte 5: 0x03, Byte 6: 0x01; the protocol version number is 1.2 and the firmware version number is 1.3.

The following information applies to both the firmware version number and the protocol version number. Backward-compatible changes result in an increment of the minor version number and backward-incompatible changes result in an increment of the major version number. The driver compares the major version number queried from the device against the major version number compiled into the driver to determine if the driver is compatible with the device. The minor version number will be reported but ignored in terms of the version-checking algorithm.

OPEN WRITE LINEAR COMMAND:

Byte 0: 0x01
Byte 1: 0x89
Byte 2 - 21: The name of the file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator
Byte 22: LSB of file size
Byte 23:
Byte 24:
Byte 25: MSB of file size

Return package:
Byte 0: 0x02
Byte 1: 0x89
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Handle: Handle number

Return packages should not be disabled for this command.

OPEN READ LINEAR COMMAND (INTERNAL COMMAND):

Byte 0: 0x01
Byte 1: 0x8A
Byte 2 - 21: The name of the file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator

Return package:
Byte 0: 0x02
Byte 1: 0x8A
Byte 2: Status: 0 equals success. Greater than 0 equal's error where the value indicates the error message.
Byte 3: LSB of pointer to linear memory segment.
Byte 4:
Byte 5:
Byte 6: MSB of pointer to linear memory segment.

Return packages should not be disabled for this command.

OPEN WRITE DATA COMMAND:

Byte 0: 0x01
Byte 1: 0x8B
Byte 2 – 21: The name of the file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator
Byte 22: LSB of file size
Byte 23:
Byte 24:
Byte 25: MSB of file size

Return package:
Byte 0: 0x02
Byte 1: 0x8B
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Handle: Handle number

Return packages should not be disabled for this command.

OPEN APPEND DATA COMMAND:

Byte 0: 0x01
Byte 1: 0x8C
Byte 2 - 21: The name of the file. Format: ASCIIZ-string with maximum size [15.3 chars] + null terminator

Return package:
Byte 0: 0x02
Byte 1: 0x8C
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Handle: Handle number
Byte 4: LSB of available file size
Byte 5: -
Byte 6: -
Byte 7: MSB of available file size

Return packages should not be disabled for this command.

BOOT COMMAND:

Byte 0: 0x01
Byte 1: 0x97
Byte 2 - 20: "Let's dance: SAMBA" + null terminator

Return package:
Byte 0: 0x02
Byte 1: 0x97
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: 'Y'
Byte 4: 'e'
Byte 5: 's'
Byte 6: '\0'

Return packages should not be disabled for this command. This command can only be accepted by USB.

SET BRICK NAME COMMAND:

Byte 0: 0x01
Byte 1: 0x98
Byte 2 – 17: New name of brick (15 characters + null terminator)

Return package:
Byte 0: 0x02
Byte 1: 0x98
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.

GET DEVICE INFO:

Byte 0: 0x01
Byte 1: 0x9B

Return package:
Byte 0: 0x02
Byte 1: 0x9B
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Bytes 3 – 17: NXT name (14 characters + null terminator)
Bytes 18-24: BT address
Byte 25: LSB of Bluetooth signal strength
Byte 26:
Byte 27:
Byte 28: MSB of Bluetooth signal strength
Byte 29: LSB of free user FLASH
Byte 30:
Byte 31:
Byte 32: MSB of free user FLASH

Return packages should not be disabled for this command.

DIRECT COMMANDS:

Byte 0: 0x00 = indication of a direct command
Byte 1 – 63: Direct command

Return package:
Byte 0: 0x02
Byte 1: “Direct command byte”
Byte 2: For now it will return a zero because the driver expects a return package for a direct command.

Please see the “LEGO MINDSTORMS NXT Direct Command” document for details on which data packages should be transmitted and received during direct commands.

All direct commands are passed on to the VM and a flag is set to indicate to the VM that new data is ready for the VM to handle.

DELETE USER FLASH:

Byte 0: 0x01
Byte 1: 0xA0

Return package:
Byte 0: 0x02
Byte 1: 0xA0
Byte 2: 0x00 (Longest time out required is approximately 3 seconds)

Return packages should not be disabled for this command.

POLL COMMAND LENGTH:

Byte 0: 0x01
Byte 1: 0xA1
Byte 2: Buffer Number: 0x00 = Poll buffer, 0x01 = High Speed buffer

Return package:
Byte 0: 0x02
Byte 1: 0xA1
Byte 2: Buffer Number: 0x00 = Poll buffer, 0x01 = High Speed buffer
Byte 3: 0 (Success)
Byte 4: Number of bytes for the command ready in the buffer (0 = no command ready). This will depend on buffer sizes.

Return packages should not be disabled for this command.

POLL COMMAND:

Byte 0: 0x01
Byte 1: 0xA2
Byte 2: Buffer Number: 0x00 = Poll buffer, 0x01 = High Speed buffer
Byte 3: Command length

Return package:
Byte 0: 0x02
Byte 1: 0xA2
Byte 2: Buffer Number: 0x00 = Poll buffer, 0x01 = High Speed buffer
Byte 3: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 4: Length of data
Byte 5-64: Command

Return packages should not be disabled for this command.

BLUETOOTH FACTORY RESET COMMAND:

Byte 0: 0x01
Byte 1: 0xA4

Return package:
Byte 0: 0x02
Byte 1: 0xA2
Byte 2: Status: 0 equals success. Greater than 0 means error where the value indicates the error message.

This command cannot be transmitted via Bluetooth because all Bluetooth functionality is reset by this command!

MESSAGE COMMAND:

A message command can be thought of as an advanced direct command. Using this command it should be possible to activate a program or a part of a program. This type of command can include another command that should be activated at the same time.

Messages are described in further detail in the LEGO MINDSTORMS NXT Direct Command document.

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:

- Success 0x00
- No more handles 0x81
- No space 0x82
- No more files 0x83
- End of file expected 0x84
- End of file 0x85
- Not a linear file 0x86
- File not found 0x87
- Handle all ready closed 0x88
- No linear space 0x89
- Undefined error 0x8A
- File is busy 0x8B
- No write buffers 0x8C
- Append not possible 0x8D
- File is full 0x8E
- File exists 0x8F
- Module not found 0x90
- Out of boundary 0x91
- Illegal file name 0x92
- Illegal handle 0x93

The following error messages are handled within the protocol layer:

- Device is not responding. This is included within the USB and Bluetooth protocols.
- Not ready for receiving data. This is included within the USB and Bluetooth protocols.

If the return package is disabled, the error message will be lost. Therefore, communication with return packages can generate problems for the host because it may believe that everything is fine when the brick has actually experienced a problem and disabled the current process.

WILD CARDS:

The following types of wild cards will be accepted within the file management system:

- Filename.Extension
- *.[File type name]
- Filename.*
- *.*

FILE NAME CONVENTIONS:

All file names within the system can include a maximum of 19 bytes (15.3 chars). File names can include uppercase and lowercase characters but will be compared within the system with the same case structure. In the user interface on the brick, only the programs files (*.rxe) will be shown as written by

the user, meaning that files with both upper and lowercases characters will be displayed with upper and lower case characters. Filename extensions will not be shown on the brick's display. Within the LEGO MINDSTORMS NXT software, it should be possible to list all file types individually or at the same time.

IO-MAP ACCESS

In order to give the advanced user more options in debugging their programs, some sort of feedback from the system should be possible. The embedded system is designed around IO-Maps, which are the well-described layer between the different controllers/drivers stack and the VM running the user's code.

The exact states and all user-accessible data are represented in the IO-Map. Using this information, the advanced user has full access and knowledge of the current state of the embedded system at runtime.

The commands in this chapter are not used directly in the LEGO MINDSTORMS NXT software!

DIRECT IO-MAP ADDRESSING AND IDENTIFICATION:

The system is FLASH-based and hence a dynamic system. The implemented modules can be of different make, complexity and version. To enable the system to find a specific module type or let the user know the programmer of a specific module, some encoding is required for the ID.

MODULE ID ENCODING:

Modules have a "unique" ID composed of the programmer's identification, a type description, and a version number including both coarse and fine version digits:

MSB **PP TT CC FF** *LSB*

The digits "PP" identify the programmer, e.g., LEGO Group = 0x01.

The digits "TT" are the module type identification.

- Cmd = 0x01
- Output = 0x02
- Input = 0x03
- Button = 0x04
- Comm = 0x05
- IOCtrl = 0x06
- Led = 0x07
- Sound = 0x08
- Loader = 0x09
- Display = 0x0A
- Low Speed = 0x0B
- UI = 0x0C

The digits "CC" and "FF" are the module version identification, e.g., 01 and 14 means version 1 release number 14.

Of course the type specified should be used generically, so differentiation between motor-driver modules should be completed by using the version as well.

REQUEST FIRST MODULE:

Byte 0: 0x01
Byte 1: 0x90
Byte 2 - 21: The resource name and extension of resource(s) to search. The extension is, of course, implicit and hence a wildcard can only be used for the module name. A wildcard is used if enumeration is the desired task; otherwise a specific name can be issued to gain information about a specific module. The name/wildcard is accompanied by (a) null byte(s) as terminator to a fixed size of 20 chars (15.3 + 0x00).

Return package:
Byte 0: 0x02
Byte 1: 0x90
Byte 2, Status: 0 (zero) returned indicating a success. Numbers > 0 indicate an error code.
Byte 3, Handle: Handle Number
Byte 4 - 23: The name of the module (resource). The name/wildcard is accompanied by (a) null byte(s) as terminator to a fixed size of 20 chars (15.3 + 0x00).
Byte 24: LSB of module I.D. (Little Endian) – 32 bit
Byte 25: -
Byte 26: -
Byte 27: MSB of module I.D.
Byte 28: LSB of module size (Little Endian) – 32 bit
Byte 29: -
Byte 30: -
Byte 31: MSB of module size
Byte 32: LSB of the module IO-MAP size (Little Endian) – 16 bit
Byte 33: MSB of module IO-MAP size

The host application should make a list of all the module IDs also containing names, sizes, etc. Further access to the modules should be gained using these IDs.

REQUEST NEXT MODULE:

Byte 0: 0x01
Byte 1: 0x91
Byte 2: Handle number from the previous Request Next Module command or from the very first Request First Module command.

Return package:
Byte 0: 0x02
Byte 1: 0x91
Byte 2, Status: 0 (zero) returned indicating a success. Numbers > 0 indicate an error code.
Byte 3, Handle: Handle Number
Byte 4 - 23: The name of the module (resource). The name/wildcard is accompanied by (a) null byte(s) as terminator to a fixed size of 20 chars (15.3 + 0x00).
Byte 24: LSB of module ID (Little Endian) – 32 bit
Byte 25: -
Byte 26: -
Byte 27: MSB of module ID
Byte 28: LSB of module size (Little Endian) – 32 bit
Byte 29: -
Byte 30: -
Byte 31: MSB of module size
Byte 32: LSB of the module IO-MAP size (Little Endian) – 16 bit
Byte 33: MSB of module IO-MAP size

CLOSE MODULE HANDLE COMMAND:

Byte 0: 0x01
Byte 1: 0x92
Byte 2: Handle number

Return package:
Byte 0: 0x02
Byte 1: 0x92
Byte 2: Status: 0 equals success. Greater than 0 means an error where the value indicates the error message.
Byte 3: Handle: Handle number

Also used for closing the IO-map handle (part of a module).

READ IO MAP COMMAND:

Byte 0: 0x01
Byte 1: 0x94
Byte 2: LSB of module ID (Little Endian)
Byte 3:
Byte 4:
Byte 5: MSB of module ID
Byte 6: LSB of offset (index) (Little Endian)
Byte 7: MSB of the offset
Byte 8: Number of bytes to be read, LSB
Byte 9: Number of bytes to be read, MSB

Return package:
Byte 0: 0x02
Byte 1: 0x94
Byte 2, Status: 0 equals success. Greater than 0 (zero) means an error where the value indicates the error message.
Byte 3: LSB of module ID
Byte 4:
Byte 5:
Byte 6: MSB of module ID
Byte 7: Number of bytes that have been read, LSB
Byte 8: Number of bytes that have been read, MSB
Byte 9 – N: IO-map contents

WRITE IO MAP COMMAND:

Byte 0: 0x01
Byte 1: 0x95
Byte 2: LSB of module ID (Little Endian)
Byte 3:
Byte 4:
Byte 5: MSB of module ID
Byte 6: LSB of offset (index) (Little Endian)
Byte 7: MSB of the offset
Byte 8: Number of bytes to be written, LSB
Byte 9: Number of bytes to be written, MSB
Byte 10 – N: IO-map content to be stored in IO-map[index]…..IO-map[index + N]

Return package:
Byte 0: 0x02
Byte 1: 0x95
Byte 2, Status: 0 equals success. Greater than 0 (zero) means an error where the value indicates the error message.
Byte 3: LSB of module ID
Byte 4: -
Byte 5: -
Byte 6: MSB of module ID
Byte 7: Number of data that have been written, LSB
Byte 8: Number of data that have been written, MSB

BLUETOOTH® COMMUNICATION

The Bluetooth® functionality within LEGO® MINDSTORMS® NXT is using the Serial Port Profile. The interface to the Bluetooth® chip in the embedded system is a high speed UART interface on the ARM7 processor. All the baseband handling is done within the Bluetooth® chip which means that the data sent to and from the Bluetooth® chip is the same data that is sent to and from the LEGO® MINDSTORMS® NXT Communication Protocol layer on the PC side.

Testing during development has shown that the Bluetooth® Serial Port communication has some disadvantages when it comes to streaming data. These disadvantages stem from two problems: bigger data packages can be received with small timing differences from the ARM processor; and there is a time penalty (of around 30 mS) within the Bluecore® chip when switching from receive-mode to transmit-mode.

To help minimizing above problems some additional data and functionality is added to the LEGO® MINDSTORMS® NXT Communication protocol during Bluetooth® communication. The additional functionality should only be used when using Bluetooth® communication.

To handle the problem of determining the size of the data that is being received, length bytes should be added in front of the LEGO® MINDSTORMS® NXT Communication Protocol payload data. These bytes describe the total length of the data packages that should be received minus the length bytes themselves. All in all, there will be two bytes describing the length of the package.

To handle the problem of the time penalty within the Bluecore™ chip, users should send data using Bluetooth® without requesting a reply package. This will mean that the Bluecore™ chip won't have to switch direction for every received package and will not incur a 30 mS penalty for every data package.

Therefore, data packages for Bluetooth® communication look as follows:

Length, LSB	Length, MSB	Command Type	Command	Byte 5	Byte 6	Etc.

Figure 3: Data packages when sending Bluetooth® commands

BLUETOOTH® CLASS OF DEVICE (COD)

The Bluetooth® chip within the LEGO® MINDSTORMS® NXT will include the following Class Of Device which can be used to determine the Bluetooth® device type:

Major Device Classes

The Major Class segment is the highest level of granularity for defining a Bluetooth® Device. The main function of a device is used to determine the major class grouping. There are 32 different possible major classes. The assignment of the major class field is defined in the table below.

Table 1: Major Device Classes

12	11	10	9	8	Major Device Class
0	0	0	0	0	Miscellaneous [Ref #2]
0	0	0	0	1	Computer (desktop, notebook, PDA, organizers, ...)
0	0	0	1	0	Phone (cellular, cordless, payphone, modem, ...)
0	0	0	1	1	LAN /Network Access point
0	0	1	0	0	Audio/Video (headset, speaker, stereo, video display, vcr, ...
0	0	1	0	1	Peripheral (mouse, joystick, keyboards, ...)
0	0	1	1	0	Imaging (printing, scanner, camera, display, ...)
0	0	1	1	1	Wearable
0	1	0	0	0	Toy
1	1	1	1	1	Uncategorized, specific device code not specified

X	X	X	X	X	All other values reserved

The LEGO® MINDSTORMS® NXT has been programmed to major device class Toy.

Minor Device Class field - Toy Major Class

The Minor Class segment is the lowest level of granularity for defining a Bluetooth® Device. There are 64 different possible minor classes. The assignment of this minor Device Class field is defined in the table below.

Table 2: Minor Device Classes

7	6	5	4	3	2	Minor Device Class bit no. of COD
0	0	0	0	0	1	Robot
0	0	0	0	1	0	Vehicle
0	0	0	0	1	1	Doll / Action Figure
0	0	0	1	0	0	Controller
0	0	0	1	0	1	Game
X	X	X	X	X	X	All other values reserved

The LEGO® MINDSTORMS® NXT has been programmed to minor device class Robot.

USB COMMUNICATION

The USB communication possibility within the LEGO® MINDSTORMS® NXT should be seen as a supplement for the users who doesn't have a Bluetooth® dongle.
Therefore the communication possibilities through the USB device should be the same through the Bluetooth® device.

Mode:	Guaranteed BW:	Payload size:	Handshake:
Bulk	No	<= 64 bytes	Yes
Isochronous	Yes	<= 1023 bytes	No
Interrupt (1.5)	No	Max. 8 bytes	Yes
Interrupt (12)	No	<= 64 bytes	Yes

Figure 4 : USB 2.0 capabilities in the Atmel ARM7 microcontroller

The USB communication is implemented using bulk transfer which means that the ARM7 processor has a 64 byte hardware buffer for receiving USB data. This limitation is also indicated within each of the command listed within the protocol when using USB communication.

Interfacing to the USB communication can be done on two different levels within the Fantom communication driver on either the PC or MAC computer. One levels which includes the above protocol commands and one which functions as a raw communication mode where the protocol parameters needs to be controlled within a higher communication layer. Having both possibilities enables using the LEGO® MINDSTORMS® NXT Protocol or writing a new protocol layer if needed. For further details please reference the documentation for the fantom driver.