## Table of Contents

1. Hardware specifications for MINDSTORMS EV3 Programmable brick ................................................................. 3
2. Communication interfaces ................................................................................................................................................. 4
3. System Command ............................................................................................................................................................... 5
   3.1 System command replies ......................................................................................................................................................... 6
   3.2 Downloading data to the EV3 programmable brick ......................................................................................................... 7
   3.2.1 File DownLoad ............................................................................................................................................................................ 8
   3.2.2 File Upload (File read) ...................................................................................................................................................... 8
   3.2.3 Directory upload .............................................................................................................................................................. 8
   3.3 System command, communication examples ............................................................................................................. 9
   3.3.1 File download ............................................................................................................................................................................. 9
   3.3.2 File Upload .............................................................................................................................................................................. 11
   3.3.3 Getting file content .............................................................................................................................................................. 13
   3.3.4 Listing files and folders .................................................................................................................................................. 15
   3.3.5 Closing file handle ............................................................................................................................................................ 17
   3.3.6 Create a directory ............................................................................................................................................................. 18
   3.3.7 Deleting a file ....................................................................................................................................................................... 19
   3.3.8 Get a list of open handles .................................................................................................................................................. 20
   3.3.9 Write to a mailbox ............................................................................................................................................................ 21
   3.3.10 Set the Bluetooth PIN code ........................................................................................................................................ 22
   3.3.11 Force the EV3 Programmable brick into Firmware update mode ..................................................... 23
4. Direct Commands ................................................................................................................................................................................. 24
   4.1 Direct Replies ................................................................................................................................................................................. 24
   4.2 Direct command, communication examples .............................................................................................................. 25
   4.2.1 Start program “Demo” on EV3 brick ........................................................................................................................................... 25
   4.2.2 Start motor B & C forward at power 50 for 3 rotation and braking at destination .................... 27
   4.2.3 Read light sensor value on sensor port 3 .......................................................................................................... 28
   4.2.4 Read the light sensor connected to port 1 as COLOR .............................................................................. 29
   4.2.5 Play a 1Kz tone at level 2 for 1 sec .........................................................................................................................30
   4.2.6 Show a picture in the display ..................................................................................................................................... 31
1  Hardware specifications for MINDSTORMS EV3 Programmable brick

LEGOMINDSTORMS EV3 programmable brick is the central processing unit within the new LEGO MINDSTORMS platform. The programmable brick consists of various advanced electronics to enable its wide range of functionalities.

Below list is a summary of the hardware specifications for the EV3 Programmable brick.

Main processor: 32-bit ARM9 processor, Texas Instrument AMI808
- 300 MHz
- OS: LINUX

Memory: 64 MB DDR RAM
16 MB FLASH
256 KB EEPROM

Micro SD-Card interface  SDHC standard, 2 – 32 GB

Bluetooth wireless communication Bluetooth V2.1 EDR, Panasonic PAN1325 module
- Texas Instrument CC2550 chip
- BlueZ Bluetooth stack
- Primary usage, Serial Port Profile (SPP)

USB 2.0 Communication, Client interface  High speed port (480 MBit/s)
USB 1.1 Communication, Host interface  Full speed port (12 MBit/s)

4 input ports  6 wire interface supporting both digital and analog interface
- Analog input 0 – 5 volt
- Support Auto-ID for external devices
  - UART communication
    o Up to 460 Kbit/s (Port 1 and 2)
    o Up to 230 Kbit/s (Port 3 and 4)

4 output ports  6 wire interface supporting input from motor encoders

Display  178x128 pixel black & white dot-matrix display
- Viewing area: 29.9 x 41.1 mm

Loudspeaker  Diameter, 23 mm

6 Buttons User interface  Surrounding UI light

Power source  6 AA batteries
- Alkaline batteries are recommended
- Rechargeable Lithium Ion battery, 2000 mAH

Connector  6-wire industry-standard connector, RJ-12 Right side adjustment
2 Communication interfaces

This section will document the protocol used for communicating between various types of masters (hosts) and the LEGO MINDSTORMS EV3 brick. The EV3 supports multiple communication interfaces: Bluetooth, USB, and WiFi. The EV3 protocol is the same for all 3 transport technologies.

Besides running user programs, the VM (virtual machine) is able to execute direct commands sent through one of the above mentioned technologies. Direct commands are composed as small programs built of regular byte codes; please reference the LEGO MINDSTORMS EV3 Firmware developer kit for more details on the individual byte codes. These direct commands (program snippets) are executed in parallel with the running user program.

Special care MUST be taken when composing these direct commands. There is NO restriction in using "dangerous" codes and constructions (e.g. Dead-locking loops in a direct command are allowed). However, a "normal" running program will continue working normal – it is only the Direct Command part of the VM which will be "dead-locked" by such a dead-locking loop.

Because of the header only containing the 2 bytes for variable allocation, direct commands are limited to one VMTHREAD only – I.e. SUBCALLs and BLOCKs is of course not possible.

Direct commands with data response can place return data in the global variable space. The global variable space is "equal to" the communication response buffer. The composition of the direct command defines at which offset the result is placed (global variable 0 is placed at offset 0 in the return buffer).

Offset in the response buffer (global variables) must be aligned (float/32 bits first and 8 bits last).

Besides direct command, the EV3 also supports system commands, which are more general terms commands which are used for downloading and upload of data to/from the embedded EV3 system.
3 System Command

#define SYSTEM_COMMAND_REPLY 0x01 // System command, reply required
#define SYSTEM_COMMAND_NO_REPLY 0x81 // System command, reply not required

System Command Bytes:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
</tr>
</thead>
</table>

Byte 0 - 1: Command size, Little Endian. Command size not including these 2 bytes

Byte 2 - 3: Message counter, Little Endian. Forth running counter

Byte 4: Command type. See defines above:

Byte 5: System Command. See the definitions below:

Byte 6 - n: Depends on the System Command given in byte 5.

System Commands:

#define BEGIN_DOWNLOAD 0x92 // Begin file download
#define CONTINUE_DOWNLOAD 0x93 // Continue file download
#define BEGIN_UPLOAD 0x94 // Begin file upload
#define CONTINUE_UPLOAD 0x95 // Continue file upload
#define BEGIN_GETFILE 0x96 // Begin get bytes from a file (while writing to the file)
#define CONTINUE_GETFILE 0x97 // Continue get byte from a file (while writing to the file)
#define CLOSE_FILEHANDLE 0x98 // Close file handle
#define LIST_FILES 0x99 // List files
#define CONTINUE_LIST_FILES 0x9A // Continue list files
#define CREATE_DIR 0x9B // Create directory
#define DELETE_FILE 0x9C // Delete
#define LIST_OPEN_HANDLES 0x9D // List handles
#define WRITEMAILBOX 0x9E // Write to mailbox
#define BLUETOOTHPIN 0x9F // Transfer trusted pin code to brick
#define ENTERFWUPDATE 0xA0 // Restart the brick in Firmware update mode
3.1 System command replies

```c
#define SYSTEM_REPLY 0x03  // System command reply OK
#define SYSTEM_REPLY_ERROR 0x05  // System command reply ERROR
```

System Reply Bytes:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
</tr>
</thead>
</table>

Byte 0 - 1: Reply size, Little Endian. Reply size not including these 2 bytes

Byte 2 - 3: Message counter, Little Endian. Equals the Direct Command

Byte 4: Reply type. See defines above

Byte 5: System Command which this is reply to.

Byte 6: System Reply Status - Error, info or success. See the definitions below:

Byte 7 - n: Further System Reply bytes depending on the System Command and the System Reply Status

SYSTEM command Reply Status codes:

```c
#define SUCCESS 0x00
#define UNKNOWN_HANDLE 0x01
#define HANDLE_NOT_READY 0x02
#define CORRUPT_FILE 0x03
#define NO_HANDLES_AVAILABLE 0x04
#define NO_PERMISSION 0x05
#define ILLEGAL_PATH 0x06
#define FILE_EXITS 0x07
#define END_OF_FILE 0x08
#define SIZE_ERROR 0x09
#define UNKNOWN_ERROR 0x0A
#define ILLEGAL_FILENAME 0x0B
#define ILLEGAL_CONNECTION 0x0C
```
3.2 Downloading data to the EV3 programmable brick

Downloading large files can be time consuming, so the download of files can be done in 2 different ways.

1. Downloading the file in largest possible chunks i.e. using the largest packet size as possible (total command size excl. Length bytes = 65534 bytes). If the total message size can be kept below 65534 bytes then all data could fit into the `begin download` command and that would be the fastest way to download that file. This is the fastest way to download but the system is also locked for this amount of time.

2. Splitting the file download into smaller portions i.e. one `begin download` followed by a number of `continue download` commands will increase the total download time, but it will also leave space (time-slice) for other commands (with higher priority) to be interleaved between the continued `continue download` commands. This is the slowest way to download files, but gives the possibility of interleaving other commands in between the `continue download` messages.

Since there is no stop or other synchronizes byte in the packets - it is essential that a message is not interrupted by other messages. I.e. when the brick has received the command size (2 first bytes of a message) ALL remaining bytes has to be transmitted and received uninterrupted. The reply (from the brick) for this very message should also be transmitted and received before any new message can be sent and processed by the brick.

The example below is build around the host application (X3 software) that wants to send a file to a P-Brick:
3.2.1 File DownLoad

- Destination filename path is addressed relative to "ims2012/sys"
- Destination folders are automatically created from filename path
- First folder name must be; "apps", "prjs" or "tools" (see ref UIdesign)
- Second folder name in filename path must be equal to byte code executable name

3.2.2 File Upload (File read)

- BEGIN UPLOAD and CONTINUE UPLOAD closes automatically the file handle when file has been uploaded.
- BEGIN GETFILE and CONTINUE GETFILE does not close the file handle when EOF has been reached
- CONTINUE GETFILE does also return the complete file size

3.2.3 Directory upload

- LIST FILES work as long as list does not exceed 1014 bytes.
3.3 System command, communication examples

In the following a sample of system command communication examples will be document to help document the interface.

3.3.1 File download

Download file ".//apps/tst/tst.rbf"

BEGIN_DOWNLOAD:

Bytes sent to brick:

```
1C00xxxx0192xxxxxxxx2E2E2F6170732F7473742F7473742E72626600
```

(bbb = bytes in message  
m = message counter  
t = type of command  
s = system command  
lllllll = file length  
nn.. = filename zero terminated)

Bytes received from brick:

```
0600xxxx03920000
```

(bbb = bytes in message  
m = message counter  
t = type of command  
s = system command  
n = return status  
hh = handle to file)
CONTINUE_DOWNLOAD:

Bytes sent to brick:

```
xxxxxxxxx819300xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
bbbmmttsshpppppppppppppppppppppppppppppppppppppppppppppppppppppp
```

- `bbbb` = bytes in message
- `mm` = message counter
- `tt` = type of command
- `ss` = system command
- `hh` = handle to file (returned in the BEGIN_DOWNLOAD)
- `pp..` = pay load

Bytes received from brick:

```
0600xxxx03930000
```

```
bbbmmttsrrhh
```

- `bbbb` = bytes in message
- `mm` = message counter
- `tt` = type of command
- `ss` = system command
- `rr` = return status
- `hh` = handle to file
3.3.2 File Upload

BEGIN UPLOAD:

Bytes send to the brick:

```
xxxxxxxxx0194xxxxxxxx
bbbbmmmttslllllnnn...
```

bbbb = bytes in message
mmmm = message counter
tt = type of command
ss = system command
llll = bytes to read
nnn... = filename incl. path

Bytes received from the brick:

```
xxxxxxxxx039400xxxxxxxx00xxx
bbbbmmmttsrrllllllllhhppp...
```

bbbb = bytes in massage
mmmm = message counter
tt = type of command
ss = system command
rr = return status
llllllll = file size
hh = file handle
ppp... = payload
CONTINUE_UPLOAD:

Bytes send to the brick:

\[070xxxx019500xxxx\]
\[bbbbmmmttsshhllll\]

\(bbbb\) = bytes in the message
\(mmmm\) = message counter
\(tt\) = type of command
\(ss\) = system command
\(hh\) = file handle
\(llll\) = bytes to read

Bytes send to the PC:

\[xxxxxxxx03950000xxx\]
\[bbbbmmmttssrrhhppp...\]

\(bbbb\) = bytes in the message
\(mmmm\) = message counter
\(tt\) = type of command
\(ss\) = system command
\(rr\) = return status
\(hh\) = handle
\(pppp..\) = payload
3.3.3 Getting file content

Used to upload datalog files - file handle is only closed when file-pointer reaches EOF and the file is not open for writing.

BEGIN_GETFILE:

Bytes send to the brick:

xxxxxxxxx0196xxxxxxxx
bbbbmmmttssllllnnn...

bbbbb = Bytes in message
mmmmm = message counter
tt = type of command
ss = system command
llll = max bytes to read
nnnn.... = path

Bytes send to the PC:

xxxxxxxxx039600xxxxxxxx00xxx
bbbbmmmttssrrllllllhhppp...

bbbbb = bytes ion message
mmmmm = message counter
tt = type of command
ss = system command
rr = return status
llllllll = file size
hh = handle
ppp... = payload
CONTINUE_GETFILE:

Bytes send to the brick:

0700xxxx019700xxxx
bbbbmmmtsshhllll

bbbb = bytes in message  
mmmm = message counter  
tt = type of command  
ss = system command  
hh = handle  
llll = max bytes to read

Bytes send to the PC:

xxxxxxxx039700xxxxxxxx00xx
bbbbmmmtssrrllllllhhppp...

bbbb = bytes in message  
mmmm = message counter  
tt = type of command  
ss = system command  
rr = return status  
llllll = file size  
hh = handle  
ppp... = payload
3.3.4 Listing files and folders

LIST_FILES:

The new line delimited list is formatted as:

If it is a file:
32 chars (hex) of MD5SUM + space + 8 chars (hex) of filesize + space + filename + new line

If it is a folder:
foldername + / + new line

Bytes send to the brick:

xxxxxxx0199xxxxxxx
bbbbmmmtssl1ll1nnn...

bbbb = bytes in message
mmmm = message counter
tt = type of message
ss = system command
llll = max. bytes to read
nnn.. = path name

Bytes send to the PC:

xxxxxxx0399xxxxxxxxxxxxxxxx
bbbbmmmtssrrllllllllhhnnn...

bbbb = bytes in message
mmmm = message counter
tt = type of message
ss = system command
rr = return status
llllll = list size
hh = handle
nnn.. = the new line delimited lists
CONTINUE_LIST_FILES:

Bytes send to the brick:

0700xxxx019Axxxxxx
bbbbmmmtsshhllll

bbbb = bytes in message
mmmm = message counter
tt = type of command
ss = system command
hh = handle
llll = max bytes to read

Bytes send to the PC:

xxxxxxxx039Axxxxxxx
bbbbmmmtssrrhhppp...

bbbb = bytes in message
mmmm = message counter
tt = type of command
ss = system command
rr = return status
hh = handle
ppp... = payload
3.3.5 Closing file handle

CLOSE_FILEHANDLE:

Bytes send to the brick:

```
xxxxxxxx019800xxxxxxxxxxxxxxxx
bbbbmmmttsshhpppppppppppppppp
```

bbbbb = bytes in the message  
mmmmm = message counter  
tt = type of message  
ss = system command  
hh = handle  
ppp... = hash

Bytes send to the PC:

```
0500xxxx039800
bbbbmmmttssrr
```

bbbbb = bytes in message  
mmmmm = message counter  
tt = type of message  
ss = system command  
rr = return status
3.3.6  Create a directory

**CREATE_DIR:**

Bytes to send to the brick:

```
xxxxxxxxx019Bxxxxxxxx...  
bbbbmmmtsspppppp...
```

*bbbb = bytes in message  
*mmmm = message counter  
*tt = type of message  
*ss = system command  
*pp = null terminated string containing full path of directory to create

Bytes send to the PC:

```
0500xxxx039Bxx  
bbbbmmmtssrr
```

*bbbb = bytes in message  
*mmmm = message counter  
*tt = type of message  
*ss = system command  
*rr = return status
3.3.7 Deleting a file

**DELETE_FILE:**

Bytes to send to the brick:

```
xxxxxxxx019Cxxxxxx...
bbbbmmmmttssppppp...
```

- `bbbb` = bytes in message
- `mmmm` = message counter
- `tt` = type of message
- `ss` = system command
- `pp` = null terminated string containing the full path of the file to delete

Bytes send to the PC:

```
0500xxxx039Cxx
bbbbmmmmttssrr
```

- `bbbb` = bytes in message
- `mmmm` = message counter
- `tt` = type of message
- `ss` = system command
- `rr` = return status
3.3.8 Get a list of open handles

LIST_OPEN_HANDLES:

Bytes to send to the brick:

`xxxxxxxx019D`
`bbbbmmmmttss`

`bbbb` = bytes in message
`mmmm` = message counter
`tt` = type of message
`ss` = system command

Bytes send to the PC:

`xxxxxxxx039Dxxxxxx....`
`bbbbmmmmttssrrpppp....`

`bbbb` = bytes in message
`mmmm` = message counter
`tt` = type of message
`ss` = system command
`rr` = return status
`pppp` = bits indicating whether handles are busy (open) or not.
3.3.9 Write to a mailbox

**WRITEMAILBOX:**

Bytes sent to another brick:

Mailbox name has to be zero terminated while the name length has to be the number of chars excluding the zero termination!

```
xxxxxxx819Exxxxxxxxxxxxxxxxxxxxx
bbbbmmmtssllaaaaa...LLLLppp...
```

bbb = bytes in the message
mmm = message counter
tt = type of message
ss = system command
ll = name Length
aaa... = name
LLLL = payload length
ppp... = payload

Reply received from another brick:

Not valid
3.3.10 Set the Bluetooth PIN code

**BLUETOOTHPIN:**

This command can only be sent by USB for safety reasons and should be formatted as:
- Bluetooth address does not contain colons
- Bluetooth MAC address is a zero terminated string type
- Bluetooth pin code is a zero terminated string type

Bytes sent to the brick:

```
0E00xxxx019F06xxxxxxxxx04xxxx
bbbbmmmtssllaaaaaaLLLLpppp
```

```
bbbb = bytes in the message
mmmm = message counter
tt = type of message
ss = system command
ll = MAC Length
aaa.. = MAC address of PC
LL = pin length
ppp... = pin code
```

Bytes send to the PC:

```
0F00xxxx039Fxx06xxxxxxxxx04xxxx
bbbbmmmtssrrllaaaaaaLLLLpppp
```

```
bbbb = bytes in message
mmmm = message counter
tt = type of message
ss = system command
rr = return status
ll = MAC length
aaa.. = MAC address of PC
LL = pin length
ppp... = pin code
```
3.3.11 Force the EV3 Programmable brick into Firmware update mode

This command is used to force the brick into Firmware update mode. The command will not send any response back to the host. The file-system will not be updated when closing (shut down) the Linux OS.

**ENTERFWUPDATE:**

Bytes send to the brick:

0400xxxx81A0
bbbbmmmttss

bbbb = bytes in massage
mmmm = message counter
tt = type of message
ss = system command
4  Direct Commands

#define DIRECT_COMMAND_REPLY 0x00  // Direct command, reply required
#define DIRECT_COMMAND_NO_REPLY 0x80  // Direct command, reply not required

Direct Command Bytes:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 0 - 1:</td>
<td>Command size, Little Endian. Command size not including these 2 bytes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 2 - 3:</td>
<td>Message counter, Little Endian. Forth running counter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 4:</td>
<td>Command type. See defines above</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 5 - 6:</td>
<td>Reservation (allocation) of global and local variables using a compressed format (globals reserved in byte 5 and the 2 lsb of byte 6, locals reserved in the upper 6 bits of byte 6) - see below:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 7 - n:</td>
<td>Byte codes as a single command or compound commands (I.e. more commands composed as a small program)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Locals = “l” and Globals = “g”

Byte 6: Byte 5:
Bit no: 76543210 76543210
Var size: llllllgg gggggggg

Global vars reservation $0 - (2^{10} - 1) 0..1023 bytes
Local vars reservation $0 - (2^{6} - 1) 0...63 bytes

4.1  Direct Replies

#define DIRECT_REPLY 0x02  // Direct command reply OK
#define DIRECT_REPLY_ERROR 0x04  // Direct command reply ERROR

Direct Reply Bytes:

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
<th>Byte 7</th>
<th>Byte 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte 0 - 1:</td>
<td>Reply size, Little Endian. Reply size not including these 2 bytes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 2 - 3:</td>
<td>Message counter, Little Endian. Equals the Direct Command</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 4:</td>
<td>Reply type. See defines above</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Byte 5 - n:</td>
<td>Response buffer. I.e. the content of the by the Command reserved global variables. I.e. if the command reserved 64 bytes, these bytes will be placed in the reply packet as the bytes 5 to 68.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.2 Direct command, communication examples

In the following a sample of direct command communication examples will be documented to help illustrate the interface more detailed. The high-level macros used are documented below.

Parameter encoding at a higher level:

To make it a bit easier to use the parameter encoding some macros defined in the "bytecodes.h" are used (See also the Parameter encoding on page 9 – 3.4 Parameter Encoding in the document “LEGO MINDSTORMS EV3 - Firmware Developer Kit” range shown encoded as signed integer:

<table>
<thead>
<tr>
<th>LCS</th>
<th>Long variable type</th>
<th>Length bytes</th>
<th>STRING zero terminated</th>
</tr>
</thead>
<tbody>
<tr>
<td>LC0(v)</td>
<td>Short constant(value)</td>
<td>single byte</td>
<td>+/- 31</td>
</tr>
<tr>
<td>LC1(v)</td>
<td>Long constant(value)</td>
<td>one byte to follow (2 bytes)</td>
<td>+/- 127</td>
</tr>
<tr>
<td>LC2(v)</td>
<td>Long constant(value)</td>
<td>two bytes to follow (3 bytes)</td>
<td>+/- 32767</td>
</tr>
<tr>
<td>LC4(v)</td>
<td>Long constant(value)</td>
<td>four bytes to follow (5 bytes)</td>
<td>+/- 2147483647</td>
</tr>
<tr>
<td>LV0(i)</td>
<td>Short LOCAL variable(adr)</td>
<td>single byte at adr</td>
<td>+/- 31</td>
</tr>
<tr>
<td>LV1(i)</td>
<td>Long LOCAL variable(adr)</td>
<td>one byte to follow at adr (2 bytes)</td>
<td>+/- 127</td>
</tr>
<tr>
<td>LV2(i)</td>
<td>Long LOCAL variable(adr)</td>
<td>two bytes to follow at adr (3 bytes)</td>
<td>+/- 32767</td>
</tr>
<tr>
<td>LV4(i)</td>
<td>Long LOCAL variable(adr)</td>
<td>four bytes to follow at adr (5 bytes)</td>
<td>+/- 2147483647</td>
</tr>
<tr>
<td>GV0(i)</td>
<td>Short GLOBAL variable(adr)</td>
<td>single byte at adr</td>
<td>+/- 31</td>
</tr>
<tr>
<td>GV1(i)</td>
<td>Long GLOBAL variable(adr)</td>
<td>one byte to follow at adr (2 bytes)</td>
<td>+/- 127</td>
</tr>
<tr>
<td>GV2(i)</td>
<td>Long GLOBAL variable(adr)</td>
<td>two bytes to follow at adr (3 bytes)</td>
<td>+/- 32767</td>
</tr>
<tr>
<td>GV4(i)</td>
<td>Long GLOBAL variable(adr)</td>
<td>four bytes to follow at adr (5 bytes)</td>
<td>+/- 2147483647</td>
</tr>
</tbody>
</table>
4.2.1 Start program “Demo” on EV3 brick

Load and run an app byte code file. This example also shows a compound direct command – i.e. two or more direct commands in one single packet. Here we load the byte code image: `"../prjs/BrkProg_SAVE/Demo.rpf"` into slot 1 – the user slot. Immediate followed by the start program in slot 1 command. Remember this is a compound command and cannot be interleaved. REMARK: The file-extension is “rpf” and NOT “rbf”. The file is a built-in “on-brick program file”.

Bytes sent to the brick:

```
opFILE,LC0(LOAD_IMAGE),LC2(USER_SLOT),LCS,'..','/','p','r','j','s','/','B','r','k','P','r','o','g','_','S','A','V','E','D','e','m','o','.','r','p','f',0,GV0(0),GV0(4),opPROGRAM_START,LC0(USER SLOT),GV0(0),GV0(4),LC0(0)
```

- `opFILE` Opcode file related
- `LC0(LOAD_IMAGE)` Command encoded as single byte constant
- `LC2(USER_SLOT)` User slot (1 = program slot) encoded as single constant byte
- `LCS` Encoding: String to follow (zero terminated)
- `"../prjs/BrkProg_SAVE/Demo.rpf"` File path and name. ".." is the “moving 1 folder up from current”
- `0x00` Zero termination of string above
- `GV0(0)` Return Image Size at Global Var offset 0. Offset encoded as single byte.
- `GV0(4)` Return Address of image at Global Var offset 4. Offset encoded as single byte.
- `opPROGRAM_START` Opcode
- `LC0(USER_SLOT)` User slot (1 = program slot) encoded as single byte constant
- `GV0(0)` Size of image at Global Var offset 0.
- `GV0(4)` Address of image at Global Var offset 4.
- `LC0(0)` Debug mode (0 = normal) encoded as single byte constant

```
30000000800800C008820100842E2F2E70726A732F42726B50726F675F534156452F44656F626C6F77646172697A65732E7465737400
```

- `30000000800800C008820100842E2F2E` = bytes in message 48 excl. packet length bytes
- `60640301606400` = message counter
- `bbbbb` = type of command - Direct command no reply
- `hhhhh` = header – variable alloc*.
- `ccccCCCCCCCC` = byte codes.

*`hhhhh = 10 least significant bits are number of globals, 6 most significial bits are locals`
4.2.2 Start motor B & C forward at power 50 for 3 rotation and braking at destination

This example uses the special OUTPUT_STEP_SPEED motor command. This command sets the speed (setpoint) for the motors in the motor-list. The command includes a ramp-up and ramp-down portion. Especially the ramp-down is useful for getting a more precise final destination. The motor brakes when the 3 rotations (3 * 360 degrees) are finished.

```
opOUTPUT_STEP_SPEED,LC0(LAYER_0),LC0(MOTOR_A + MOTOR_B),LC1(SPEED_50),LC0(0),LC2(900), LC2(180),LC0(BRAKE)
```

- **Opcode**
- **LC0(0)** Layer 0 – encoded as single byte constant
- **LC0(MOTOR_A + MOTOR_B)** Motor B & C (motor list) encoded as single byte constant
- **LC1(SPEED_50)** Speed 50% encoded as one constant byte to follow
- **LC0(0)** No STEP1 i.e. full speed from beginning – encoded as single byte constant.
- **LC2(900)** STEP2 for 2.5 rotation (900 degrees) – encodes as two bytes to follow.
- **LC2(180)** STEP3 for 0.5 rotation (180 degrees) for better precision at destination – encoded as two bytes to follow.
- **LC0(BRAKE)** Brake (1) – encoded as single byte constant.

Bytes sent to the brick:

```
1200xxxx800000AE00068132008240382840001
Bbbaaaaaaattttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
4.2.3 Read light sensor value on sensor port 3

This direct command will read the light-sensor connected to the input port 3 on the brick. The mode is explicitly set to mode 0 (zero) i.e. the native mode 0 for a Light Sensor (0 – 100 pct.). The returned value is a 32 bit float encoded as SI 0-100 pct.

Default 32 bit float (SI 0-100 pct.)

```
oINPUT_DEVICE,LC0(READY_SI),LC0(LAYER_0),LC0(SENSOR_PORT_3),LC0(DO_NOT_CHANGE_TYPE),
LC0(MODE_0),LC0(ONE_DATA_SET),LC0(GLOBAL_VAR_INDEX0)
```

- `opINPUT_DEVICE` Opcode input related
- `LC0(READY_SI)` Command (READY_SI) encoded as single byte constant
- `LC0(LAYER_0)` Layer number (0 = this very brick) encoded as single byte constant
- `LC0(SENSOR_PORT_3)` Sensor connected to port 3 (1-4 / 0-3 internal) encoded as single byte constant
- `LC0(DO_NOT_CHANGE_TYPE)` If set to 0 (zero) = don’t change type - encoded as single byte constant
- `LC0(MODE_0)` Mode 0 - encoded as single byte constant
- `LC0(ONE_DATA_SET)` Count of datasets (Mode 0 has only 1 (pct)) - encoded as single byte constant
- `LC0(GLOBAL_VAR_INDEX0)` Place returned value in Global var at index 0 (zero) - encoded as single byte constant

Bytes sent to the brick:

```
0D00xxxx000400991D000200000160
BbbmmmtthhhhCCCCCCCCCCCCCCCC
```

- `bbbb = bytes in message 13 excl. packet length bytes`
- `mmmm = message counter`
- `tt = type of command - Direct command with reply`
- `hhhh = header – variable alloc. Here 4 bytes reserve in Global Vars`.
- `CC/cc/CC/cc = byte codes.

1 `hhhh = 10 least significant bits are number of globals, 6 most significal bits are locals`
4.2.4  Read the light sensor connected to port 1 as COLOR

This direct command will read the light-sensor connected to the input port 1 on the brick. The mode is explicitly set to mode 2 “COLOR mode”. The sensor will return a value between 0-8 (both included) i.e. the color of the object in front of the sensor. The returned value is a 32 bit float encoded as 0-8.

opINPUT_DEVICE, LC0(READY_SI), LC0(LAYER_0), LC0(SENSOR_PORT_1), LC0(DO_NOT_CHANGE_TYPE), LC0(MODE_2), LC0(ONE_DATA_SET), LC0(GLOBAL_VAR_INDEX0)

Bytes sent to the brick:

8D00xxxx000400991D000000020160
BbbbbmmmtthhhhhCCCCCCCCCCCCCCCC

bbbb = bytes in message 13 excl. packet length bytes
mmmm = message counter
tt = type of command - Direct command with reply
hhhh = header – variable alloc. Here 1 byte reserve in Global Vars*).
CC/cc/CC/cc = byte codes.

*hhhh = 10 least significant bits are number of globals, 6 most significal bits are locals
4.2.5 Play a 1KHz tone at level 2 for 1 sec.

```
opSOUND, LC0(TONE), LC1(2), LC2(1000), LC2(1000)
```

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>opSOUND</td>
<td>Opcode sound related</td>
</tr>
<tr>
<td>LC0(TONE)</td>
<td>Command (TONE) encoded as single byte constant</td>
</tr>
<tr>
<td>LC1(2)</td>
<td>Sound-level 2 encoded as one constant byte to follow</td>
</tr>
<tr>
<td>LC2(1000)</td>
<td>Frequency 1000 Hz. encoded as two constant bytes to follow</td>
</tr>
<tr>
<td>LC2(1000)</td>
<td>Duration 1000 mS. encoded as two constant bytes to follow</td>
</tr>
</tbody>
</table>

Bytes sent to the brick:

```
0F00xxxx80000094010282E80382E803
Bbbbmmmtthhhcccccccccccccccccccc
```

- `bbbb` = bytes in message 15 excl. packet length bytes
- `mmmm` = message counter
- `tt` = type of command - Direct command no reply
- `hhhh` = header – variable alloc\(^7\)
- `cc/CC` = byte codes.

\(^7\) `hhhh` = 10 least significant bits are number of globals, 6 most significant bits are locals
4.2.6 Show a picture in the display

Clears the screen and draws the bmp-image “mindstorms.rgf” on the display at the coordinates \((x = 0, y = 50)\). First the screen is cleared by the FILLWINDOW sub-command, then the bmp-image file is loaded by the sub-command BMPFILE. Nothing happens on the screen before the UPDATE sub-command is issued.

\[
\text{opUI\_DRAW, LC0(FILLWINDOW), LC0(BG\_COLOR), LC2(\theta), LC2(\theta), opUI\_DRAW, LC0(BMPFILE),}
\]
\[
\text{LC0(FG\_COLOR), LC2(\theta), LC2(50), LCS, 'u', 'i', '/' , 'm', 'i', 'n', 'd', 's', 't', 'o', 'r', 'm', 's', . , 'r', 'g', 'f', 0, opUI\_DRAW, LC0(UPDATE)}
\]

Bytes sent to the brick:

\[
2C00000000800008413008200082000841C01820008232008475692F6D696E6473746FD76D7B
\]
\[
32E72676008400  
ccccccccccCCC
\]

\[
\text{biii = bytes in message 44 excl. packet length bytes} 
\text{mmmm = message counter} 
\text{tt = type of command - Direct command no reply} 
\text{hhhh = header – variable alloc¹} 
\text{CC/cc/CC/cc = byte codes.} 
\]

¹\text{hhhh = 10 least significant bits are number of globals, 6 most significant bits are locals}