Firmware Developer Kit

LEGO® MINDSTORMS® EV3 Firmware Developer Kit

Table of Contents

1 Hardware specifications for MINDSTORMS EV3 Programmable brick

LEGO MINDSTORMS 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 AM1808 - 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 Firmware architecture used within MINDSTORMS EV3

The firmware architecture within LEGO MINDSTORMS EV3 is build around a LINUX operating system. Multiple shared libraries are developed each enabling various functionality which are utilized by the LEGO application (Virtual machine), which again are controlled through various system programs and user programs (Byte-code programs).

Figure 1 below provides a graphical representation of the firmware architecture from a high level perspective.

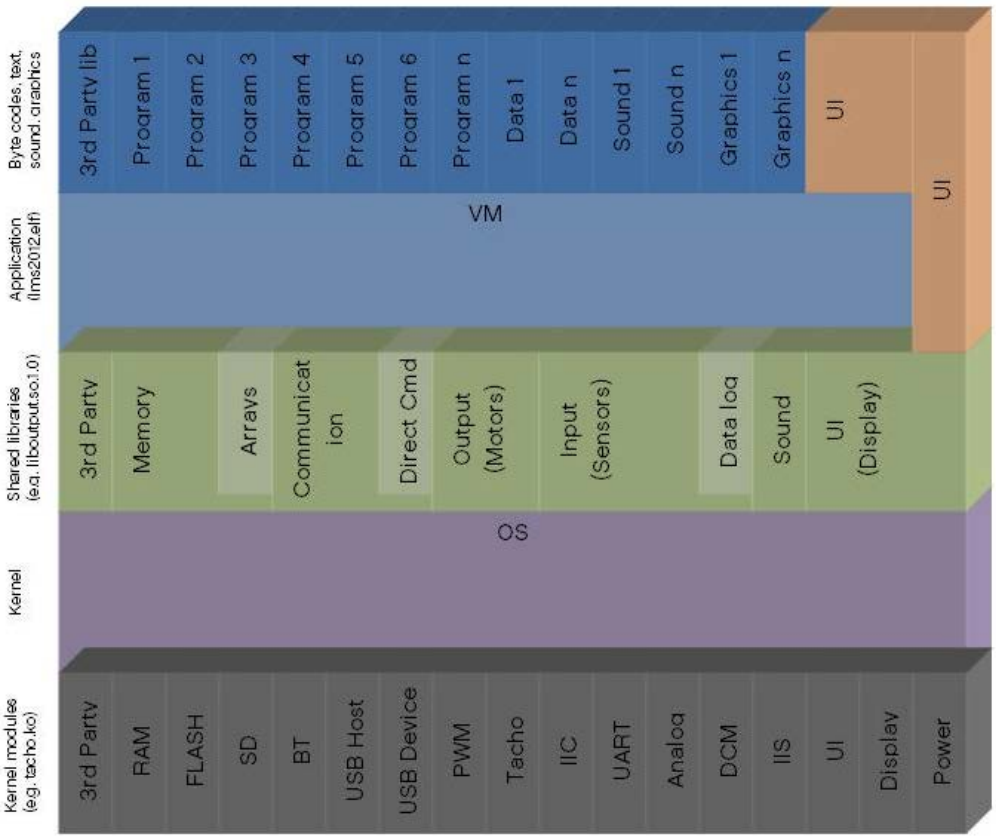


Figure 1 : Graphical representation of the firmware architecture

2.1 Kernel modules

The kernel modules handles all initialization and control of hardware components both internal within the ARM9 controller and towards external hardware components. Most of these drivers will be interrupt driven. The kernel modules are represented as the black layer within Figure 1.

2.2 Operating system (OS)

MINDSTORMS EV3 is build around linux platform support packages (PSP) for AM1808 ARM9 controller. The specific packages used DaVinci-PSP-SDK-03.20.00.13 utilizes a LINUX kernel version 2.6.33 RC4. The actual used operating system is compiled using “GNU + CodeSourcery__G++ lite” compilers with various special settings.

2.3 Shared libraries

The system will consist of multiple shared libraries which individually handle various specific functionalities. The shared libraries are represented as the green layer within Figure 1.
These shared libraries are implemented as "Asynchronous driver model" which mean that polling or direct signalling are needed between the shared library and virtual machine.

The functionality implemented within the shared libraries will be accessed from the virtual machine through system calls from the virtual machine into the shared libraries. All system commands are implemented in a way that is none blocking.

The shared libraries are implemented in C and compiled into native code to enable the optimal performance. New functionality can be added to the firmware through either improving existing shared libraries or developing new shared libraries to the system.

Initially the firmware includes the following shared libraries:

Shared libraries within LEGO MINDSTORMS EV3 Firmware	
Input library	Handle all functionality around input ports
Output library	Handle all functionality around output ports
Sound library	Handle all functionality around playing sounds
User interface library	Handle all functionality around user interface
Communication library	Handle all communication with EV3 P-brick (USB, Bluetooth, WiFi)
Memory library	Handle all functionality around memory management

2.4 Virtual machine

The virtual machine should be thought of as the actual application running on the EV3 P-brick. The virtual machine handles the actual program execution and calls into the shared libraries (Functionality device drivers). The virtual machine is represented as the blue layer within Figure 1.
Program execution is done through byte-code which represents an abstraction of various functionalities within the shared libraries. These byte-codes will be documented in more detailed within section 4.

The virtual machine is implemented using a memory based architecture in order to match the output from the compiler most optimal.

New functionality can be achieved through implementation of sub-routines within programs. This approach enables for example 3rd party developers to implement new functionality without having to develop new shared libraries and build new firmware versions.
All user functionality needed when the products launched is implemented within shared libraries, with the exception of the user interface which is launched when the LEGO MINDSTORMS EV3 P-brick starts. This interface is implemented using byte-codes as this enables updating the user interface without having to rebuild the entire firmware.

A few main characteristic for the virtual machine:

- The following data-types are supported:
  - 8-bit integer
  - 16-bit integer
  - 32-bit integer
  - Float (IEEE 754 single precision – 32 bits)
  - Strings
  - Arrays (Implemented as shared library functionality)
- There is no support for polymorphism (Data types should be defined at compile time)

3 Program structure

This section will outline the structure required within user programs which the virtual machine is able to execute. Basically a program consists of the following three different components in the mentioned fixed order:

- Image header
- Object headers
- Byte-codes

Little Endian (addresses and data are represented with LSB on lowest address and MSB on highest address) is expected throughout the entire program. Referencing offset to instructions is number of bytes from start of image to start of object instructions. Index to global variables are byte based and counted from start of globals (zero based) while index to local variables are byte based and counted from start of object locals (zero based). Object ID's is not zero based - First object (VMTHEAD) is named 1.

3.1 Image header

The image header documents image version, filesize, no of objectheaders (objects) and no of global variable bytes.

Image header	
Sign	4 bytes
Image size	4 bytes
Version info	2 bytes
Number of objects	2 bytes
Number of global bytes	4 bytes

3.2 Object header

Objectheaders contains different informations depending on the nature of the object:

- The VMTHREAD object (former TBC__TOPVI)
  - OffsetToInstructions tells were to find the corresponding byte codes (offset from image start)
  - OwnerObjectId must be zero
  - TriggerCount is used but must be zero
  - LocalBytes describes the number of bytes for local variables

- The SUBCALL object (former TBC__VI and TBC__VI__ALIAS)
  - OffsetToInstructions tells were to find the corresponding byte codes (if alias this is equal to mother object)
  - OwnerObjectId must be zero
  - TriggerCount is used and must be one
  - LocalBytes describes the number of bytes for local variables

- The BLOCK object (former CLUMP)
  - OffsetToInstructions tells were to find the corresponding byte codes (offset from image start)
  - OwnerObjectId is equal to object id it belongs to (not equal to zero)
  - TriggerCount is used to determine how many triggers needed before the BLOCK object is activated
  - LocalBytes must be zero (locals are defined in the owner object)

Object header	
Offset to instructions	4 bytes
Owner object id	2 bytes
Trigger count	2 bytes
Local bytes	4 bytes

3.3 Bytecodes

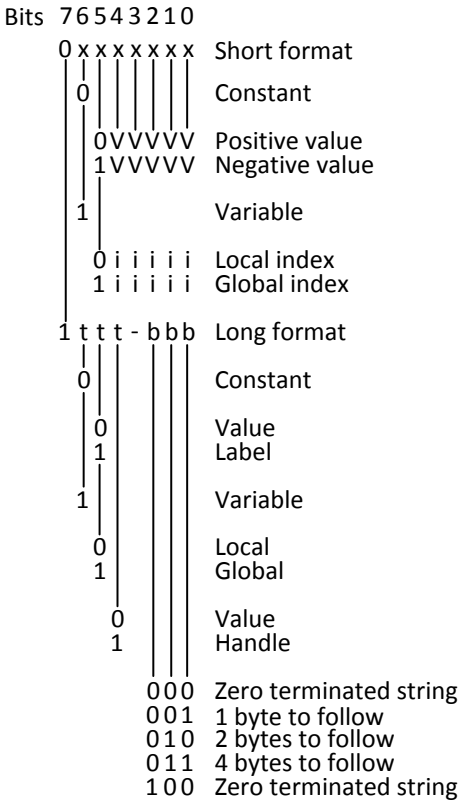
The individual byte codes will be documented in details within section 0. Below are some general guidelines for how the byte codes are implemented.

- Little Endian is used
  - Addresses and data are represented with LSB on lowest address and MSB on highest address
- Offset to instructions is number of bytes from start of image to start of object instructions
- Index to global variables is byte based and counted from start of globals (zero based)
- Index to local variables is byte based and counted from start of object locals (zero based)
- Object ID's is not zero based - First object (VMTHEAD) is named 1
- FILE layout needs to be aligned

3.4 Parameter encoding

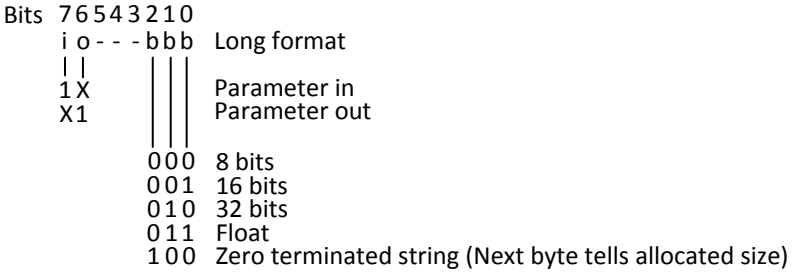
Parameter types and values for primitives, system calls and subroutine calls are encoded in the callers byte code stream as follows:

opAdd8 (ParCode1, ParCode2, ParCodeN)



For subroutine parameters additional information about number of parameters, direction and size are needed and placed in front of the called code.
Parameters MUST be sorted as follows: Largest (4 Bytes) parameters is first and smallets (1 Byte) is last in the list.

OffsetToInstructions => NoOfPars, ParType1, ParType2, ParTypeN



4 Byte code definition and functionality

This section will document the individual up-codes which the virtual machine is able to execute.

4.1 General operations

Instruction	**OpError**
Opcode	0x00
Arguments	
Dispatch status	Changed to INSTRBREAK
Description	Instruction can be used to indicate error. This code is not normally used in programs.

Instruction	**OpNop**
Opcode	0x01
Arguments	
Dispatch status	Unchanged
Description	Instruction will do nothing.

Instruction	**OpProgram_Stop (PRGID)**
Opcode	0x02
Arguments	(Data16) Program ID slot
Dispatch status	Changed to PRGBREAK
Description	Stop specific program ID slot

The following program ID definition can be referenced:

0 Program slot served for executing the user interface
1 Program slot used for executing user programs
2 Program slot used for direct commands received from external
3 Program slot used for direct commands coming from the user interface
4 Program slot used for debug user interface
-1 Current slot

Instruction	**OpProgram_Start (PRGID, SIZE, *IP, DEBUG)**
Opcode	0x03
Arguments	(Data16) Program ID slot (Data32) Size of image (Data32) Address of image (Value obtained from opFILE(LOAD_Image,..)) (Data8) Debug mode
Dispatch status	Unchanged
Description	This function is to be used for starting an existing user program on the EV3 P-brick.

The debug mode parameter can take the following values:

0 Normal mode
1 Debug mode, Requires debugging tools to be installed and running
2 Do not execute the program

Instruction	**OpObject_Stop (OBJID)**
Opcode	0x04
Arguments	(Data16) OBJID
Dispatch status	Changed to STOPBREAK
Description	This function can be used for stopping a specific object in further execution.

Instruction	**opObject_Start (OBJID)**
Opcode	0x05
Arguments	(Data16) OBJID
Dispatch status	Unchanged
Description	This function can be used for starting a specific object to execute

Instruction	**opObject_Trig (OBJID)**
Opcode	0x06
Arguments	(Data16) OBJID
Dispatch status	Unchanged
Description	This function trigger the specific object and initaite execution if alle trigger requirements are active.

Instruction	**opObject_Wait (OBJID)**
Opcode	0x07
Arguments	(Data16) OBJID
Dispatch status	Status can change to BUSYBREAK if object is not finalized
Description	This function will make the specific execution wait until the specific object have finalized

Instruction	**opReturn**
Opcode	0x08
Arguments	
Dispatch status	Status changed to STOPBREAK
Description	This function enables returning from subroutine

Instruction	**opCall (OBJID, PARAMETERS)**
Opcode	0x09
Arguments	(Data16) OBJID (Data8) PARAMETERS, please reference parameter encoding section 0
Dispatch status	Status changed to STOPBREAK or BUSYBREAK
Description	This function can be used for calling sub-routines

Instruction	**opObject_End**
Opcode	0x0A
Arguments	
Dispatch status	Status changed to STOPBREAK
Description	This function should be used in the end of object to indicated the end of the object

Instruction	**opSleep**
Opcode	0x0B
Arguments	
Dispatch status	Status changed to INSTRBREAK
Description	This function will break current VM-thread execution

Instruction	**opProgram_Info (CMD, PRGID, DATA)**
Opcode	0x0C
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Status changed to FAILBREAK
Description	Depending on selected command type

CMD: OBJ_STOP = 0x00
Arguments
(Data16) PRGID – Program slot number
(Data16) OBJID – Object id

CMD: OBJ_START = 0x04
Arguments
(Data16) PRGID – Program slot number
(Data16) OBJID – Object id

CMD: GET_STATUS = 0x16
Arguments
(Data16) PRGID – Program slot number
Returns
(Data8) Data – Program status

CMD: GET_ SPEED = 0x17
Arguments
(Data16) PRGID – Program slot number
Returns
(Data32) Data – Program speed [instr/S]

CMD: GET_ PRGRESULT = 0x18
Arguments
(Data16) PRGID – Program slot number
Returns
(Data8) Data – Program result [OK, BUSY, FAIL]

Instruction	**opLabel (NO)**
Opcode	0x0D
Arguments	(Data8) NO – Label number
Dispatch status	Unchanged
Description	This function enables marking an address with a label

Instruction	**opProbe (PRGID, OBJID, OFFSET, SIZE)**
Opcode	0x0E
Arguments	(Data16) PRGID – Program slot number (Data16) OBJID – Object id (Zero means globals) (Data32) OFFSET – Offset (Start from…) (Data32) SIZE – Size (length of dump) zero means all (max 1024)
Dispatch status	Unchanged
Description	This function enables displaying global's or object locals on the terminal

Instruction	**opDo (PRGID, IMAGE)**
Opcode	0x0F
Arguments	(Data16) PRGID – Program slot number (Data32) *IMAGE – Address of image (Data32) *GLOBAL – Address of global variables
Displatch status	Status changed to BUSYBREAK
Description	This function enables activating a small byte code snippet

Instruction	**opBP0**
Opcode	0x88
Arguments	
Dispatch status	Unchanged
Description	This function enables displaying global or objects locals on terminal when debug is enabled. Removes itself when done.

Instruction	**opBP1**
Opcode	0x89
Arguments	
Dispatch status	Unchanged
Description	This function enables displaying global or objects locals on terminal when debug is enabled. Removes itself when done.

Instruction	**opBP2**
Opcode	0x8A
Arguments	
Dispatch status	Unchanged
Description	This function enables displaying global or objects locals on terminal when debug is enabled. Removes itself when done.

Instruction	**opBP3**
Opcode	0x8B
Arguments	
Dispatch status	Unchanged
Description	This function enables displaying global or objects locals on terminal when debug is enabled. Removes itself when done.

Instruction	**opPB_Set (PRGID, NO, ADDRESS)**
Opcode	0x8C
Arguments	(Data16) PRGID – Program slot number
	(Data8) NO – Breakpoint number [0-3]
	(Data32) ADDRESS – Address (Offset from start of image) (0 = Remove breakpoint)
Dispatch status	Unchanged
Description	This function enables setting a breakpoint within byte code program

Instruction	**opRandom (MIN, MAX, VALUE)**
Opcode	0x8E
Arguments	(Data16) MIN – Minimum value
	(Data16) MAX – Maximum value
Returns	
	(Data16) VALUE – Random value within the min and max range
Dispatch status	Unchanged
Description	This function enables getting a random value within the specified range

Instruction	**opInfo (CMD, ...)**
Opcode	0x7C
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	Information on functions entry

CMD: SET_ERROR = 0x01
Arguments
(Data8) NUMBER – Error number

CMD: GET_ERROR = 0x02
Arguments
(Data8) NUMBER – Error number

CMD: ERRORTEXT = 0x03
Arguments
(Data8) NUMBER – Error number
(Data8) LENGTH – Maximum length of string returned (-1 = No check)
Returns
(Data8) DESTINATION – String variable or handle to string
Description
Convert error number to text string

CMD: GET_VOLUME = 0x04
Returns
(Data8) VALUE – Volume [0-100%]

CMD: SET_VOLUME = 0x05
Arguments
(Data8) VALUE – Volume [0-100%]

CMD: GET_MINUTES = 0x06
Returns
(Data8) VALUE – Minutes to sleep [0-120 min]

CMD: SET_MINUTES = 0x07
Arguments
(Data8) VALUE – Minutes to sleep [0-120 min]

Instruction	**opString (CMD, ...)**
Opcode	0x7D
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	String function entry

CMD: GET__SIZE = 0x01
Arguments
(Data8) SOURCE1 – String variable or handle to string
Returns
(Data16) SIZE – Size of string
Description
Get size of string, not including zero termination

CMD: ADD = 0x02
Arguments
(Data8) SOURCE1 – String variable or handle to string
(Data8) SOURCE2 – String variable or handle to string
Returns
(Data8) DESTINATION – String variable or handle to string

CMD: COMPARE = 0x03
Arguments
(Data8) SOURCE1 – String variable or handle to string
(Data8) SOURCE2 – String variable or handle to string
Returns
(Data8) RESULT – Result (0 = Not equal, 1 = equal)

CMD: DUPLICATE = 0x05
Arguments
(Data8) SOURCE1 – String variable or handle to string
Returns
(Data8) DESTINATION – String variable or handle to string

CMD: VALUE__TO__STRING = 0x06
Arguments
(DataF) VALUE – Value which are to be converted
(Data8) FIGURES – Total number of figures inclusive decimal point
(Data8) DECIMALS – Number og decimals
Returns
(Data8) DESTINATION – String variable or handle to string
Description
Convert floating point value to a string

CMD: STRING_TO_VALUE = 0x07
Arguments
(Data8) SOURCE1 – String variable or handle to string
Returns
(DataF) Value – Floating point value
Description
Convert string to floating point value

CMD: STRIP = 0x08
Arguments
(Data8) SOURCE1 – String variable or handle to string
Returns
(Data8) DESTINATION – String variable or handle to string
Description
Strip a string for spaces

CMD: NUMBER_TO_STRING = 0x09
Arguments
(Data16) VALUE – Value which are to be converted
(Data8) FIGURES – Total number of figures inclusive decimal point
Returns
(Data8) DESTINATION – String variable or handle to string

CMD: SUB = 0x0A
Arguments
(Data8) SOURCE1 – String variable or handle to string
(Data8) SOURCE2 – String variable or handle to string
Returns
(Data8) DESTINATION – String variable or handle to string
Description
Returns a substring from source 1 that starts were source 2 ends

CMD: VALUE_FORMATTED = 0x0B
Arguments
(DataF) VALUE – Value which are to be converted
(Data8) FORMAT – Format string variable or handle to string
(Data8) SIZE – Total size of destination string
Returns
(Data8) DESTINATION – String variable or handle to string
Description
Convert floating point value to formatted string

CMD: NUMBER_FORMATTED = 0x0C

Arguments

(Data32) NUMBER – Value which are to be converted

(Data8) FORMAT – Format string variable or handle to string

(Data8) SIZE – Total size of destination string

Returns

(Data8) DESTINATION – String variable or handle to string

Description

Convert integer value to formatted string

Instruction	**opMemory_Write (PRGID, OBJID, OFFSET, SIZE, ARRAY)**
Opcode	0x7E
Arguments	(Data16) PRGID – Program slot number (Data16) OBJID – Object id (Zero means globals) (Data32) OFFSET – Offset (Start from…) (Data32) SIZE – Length of array to write (Data8) ARRAY – First element of data8 array to write
Dispatch status	Unchanged
Description	This function enables writting directly into program memory

Instruction	**opMemory_Read (PRGID, OBJID, OFFSET, SIZE, ARRAY)**
Opcode	0x7F
Arguments	(Data16) PRGID – Program slot number (Data16) OBJID – Object id (Zero means globals) (Data32) OFFSET – Offset (Start from…) (Data32) SIZE – Length of array to read
Returns	(Data8) ARRAY – First element of data8 array to receive
Dispatch status	Unchanged
Description	This function enables writting directly into program memory

Instruction	**opPort_Cnv_Output (PortIn, Layer, Bitfield, Inverted)**
Opcode	0x61
Arguments	(Data32) PortIn – Encoded port number
Returns	(Data8) Layer – Value indicating layer number (Data8) Bitfield – Bitfield indicating specific port(s) (Data8) Inverted – True if left/right motor are inverted
Dispatch status	Unchanged
Description	This function enables converter port to layer and bitfield used on output ports

Instruction	**opPort_Cnv_Input (PortIn, Layer, PortOut)**
Opcode	0x61
Arguments	(Data32) PortIn – Encoded port number
Returns	(Data8) Layer – Value indicating layer number (Data8) PortOut – Indicated the relevant port to use (Zero adjusted)
Dispatch status	Unchanged
Description	This function enables converter port to layer and bitfield used on input ports

Instruction	**opNote_To_Freq (NOTE, FREQ)**
Opcode	0x63
Arguments	(Data8) NOTE – Note string (HND) (E.C. ”C#4”)
Return	(Data16) FREQ – Frequency [Hz]
Dispatch status	Unchanged
Description	This function enables converting note to a tone

Instruction	**opSystem (COMMAND, STATUS)**
Opcode	0x60
Arguments	(Data8) COMMAND – Command string (HND)
Return	(Data32) STATUS – Status of command
Dispatch status	Unchanged
Description	This function enables executing a system command string (HND)

4.2 Math instructions

Instruction	**opAdd8 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x10
Arguments	(Data8) SOURCE1 – Value 1 (Data8) SOURCE2 – Value 2
Return	(Data8) DESTINATION – Value = Value 1 + Value 2
Dispatch status	Unchanged
Description	This function enables adding two 8-bit values together

Instruction	**opAdd16 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x11
Arguments	(Data16) SOURCE1 – Source 1 value / index (Data16) SOURCE2 – Source 2 value / index
Return	(Data16) DESTINATION – Value = Value 1 + Value 2
Dispatch status	Unchanged
Description	This function enables adding two 16-bit values together

Instruction	**opAdd32 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x12
Arguments	(Data32) SOURCE1 – Source 1 value / index (Data32) SOURCE2 – Source 2 value / index
Return	(Data32) DESTINATION – Value = Value 1 + Value 2
Dispatch status	Unchanged
Description	This function enables adding two 32-bit values together

Instruction	**opAddF (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x13
Arguments	(DataF) SOURCE1 – Source 1 value / index (DataF) SOURCE2 – Source 2 value / index
Return	(DataF) DESTINATION – Value = Value 1 + Value 2
Dispatch status	Unchanged
Description	This function enables adding two floating point values together

Instruction	**opSub8 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x14
Arguments	(Data8) SOURCE1 – Source 1 value / index (Data8) SOURCE2 – Source 2 value / index
Return	(Data8) DESTINATION – Value = Value 1 - Value 2
Dispatch status	Unchanged
Description	This function enables subtracting one 8-bit value from another 8-bit value

Instruction	**opSub16 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x15
Arguments	(Data16) SOURCE1 – Source 1 value / index (Data16) SOURCE2 – Source 2 value / index
Return	(Data16) DESTINATION – Value = Value 1 - Value 2
Dispatch status	Unchanged
Description	This function enables subtracting one 16-bit value from another 16-bit value

Instruction	**opSub32 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x16
Arguments	(Data32) SOURCE1 – Source 1 value / index (Data32) SOURCE2 – Source 2 value / index
Return	(Data32) DESTINATION – Value = Value 1 - Value 2
Dispatch status	Unchanged
Description	This function enables subtracting one 32-bit value from another 32-bit value

Instruction	**opSubF (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x17
Arguments	(DataF) SOURCE1 – Source 1 value / index (DataF) SOURCE2 – Source 2 value / index
Return	(DataF) DESTINATION – Value = Value 1 - Value 2
Dispatch status	Unchanged
Description	This function enables subtracting one floating point value from another floating point value

Instruction	**opMul8 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x18
Arguments	(Data8) SOURCE1 – Source 1 value / index (Data8) SOURCE2 – Source 2 value / index
Return	(Data8) DESTINATION – Value = Value 1 * Value 2
Dispatch status	Unchanged
Description	This function enables multiplying two 8-bit values together

Instruction	**opMul16 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x19
Arguments	(Data16) SOURCE1 – Source 1 value / index (Data16) SOURCE2 – Source 2 value / index
Return	(Data16) DESTINATION – Value = Value 1 * Value 2
Dispatch status	Unchanged
Description	This function enables multiplying two 16-bit values together

Instruction	**opMul32 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x1A
Arguments	(Data32) SOURCE1 – Source 1 value / index (Data32) SOURCE2 – Source 2 value / index
Return	(Data32) DESTINATION – Value = Value 1 * Value 2
Dispatch status	Unchanged
Description	This function enables multiplying two 32-bit values together

Instruction	**opMulF (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x1B
Arguments	(DataF) SOURCE1 – Source 1 value / index (DataF) SOURCE2 – Source 2 value / index
Return	(DataF) DESTINATION – Value = Value 1 * Value 2
Dispatch status	Unchanged
Description	This function enables multiplying two floating points together

Instruction	**opDiv8 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x1C
Arguments	(Data8) SOURCE1 – Source 1 value / index (Data8) SOURCE2 – Source 2 value / index
Return	(Data8) DESTINATION – Value = Value 1 / Value 2
Dispatch status	Unchanged
Description	This function enables diving one 8 bit value with another 8 bit value

Instruction	**opDiv16 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x1D
Arguments	(Data16) SOURCE1 – Source 1 value / index (Data16) SOURCE2 – Source 2 value / index
Return	(Data16) DESTINATION – Value = Value 1 / Value 2
Dispatch status	Unchanged
Description	This function enables diving one 16 bit value with another 16 bit value

Instruction	**opDiv32 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x1E
Arguments	(Data32) SOURCE1 – Source 1 value / index (Data32) SOURCE2 – Source 2 value / index
Return	(Data32) DESTINATION – Value = Value 1 / Value 2
Dispatch status	Unchanged
Description	This function enables diving one 32 bit value with another 32 bit value

Instruction	**opDivF (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x1F
Arguments	(DataF) SOURCE1 – Source 1 value / index (DataF) SOURCE2 – Source 2 value / index
Return	(DataF) DESTINATION – Value = Value 1 / Value 2
Dispatch status	Unchanged
Description	This function enables diving one floating point value with another floating point value

Instruction	**opMath (CMD, ...)**

Opcode	0x8D
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	Math function entry

CMD: EXP = 0x01
Arguments
(DataF) DATA X – Exponent value
Returns
(DataF) RESULT – Calculated value
Description
e^X, Result = expf(X)

CMD: MOD = 0x02
Arguments
(DataF) DATA X – Division numerator
(DataF) DATA Y – Division denominator
Returns
(DataF) RESULT – String variable or handle to string
Description
Modulo, Result = fmod(x,y)

CMD: FLOOR = 0x03
Arguments
(DataF) DATA X – Floating point input number
Returns
(DataF) RESULT – Largest real number not greater than X
Description
Result = floor(x)

CMD: CEIL = 0x04
Arguments
(DataF) DATA X – Floating point input number
Returns
(DataF) RESULT – Smallest real number not less than X
Description
Result = ceil(x)

CMD: ROUND = 0x05
Arguments
(DataF) DATA X – Floating point input number
Returns
(DataF) RESULT – Rounded real value
Description
Result = round(x)

CMD: ABS = 0x06

Arguments

(DataF) DATA X – Floating point input number

Returns

(DataF) RESULT – Absolute value

Description

Result = fabs(x)

CMD: NEGATE = 0x07

Arguments

(DataF) DATA X – Floating point input number

Returns

(DataF) RESULT – Input value negated

Description

Result = 0.0 - x

CMD: SQRT = 0x08

Arguments

(DataF) DATA X – Floating point input number

Returns

(DataF) RESULT – Square root of input value

Description

Square root, Result = sqrt(x)

CMD: LOG = 0x09

Arguments

(DataF) DATA X – Floating point input number

Returns

(DataF) RESULT – 10 logarithm of input value

Description

Result = log10(x)

CMD: Ln = 0x0A

Arguments

(DataF) DATA X – Floating point input number

Returns

(DataF) RESULT – Logarithm of input value

Description

Result = log(x)

CMD: SIN = 0x0B

Arguments

(DataF) DATA X – Floating point input number

Returns

(DataF) RESULT – Return sinus geometry value

CMD: COS = 0x0C

Arguments
(DataF) DATA X – Floating point input number
Returns
(DataF) RESULT – Return cosine geometry value

CMD: TAN = 0x0D
Arguments
(DataF) DATA X – Floating point input number
Returns
(DataF) RESULT – Return tangent geometry value

CMD: ASIN = 0x0E
Arguments
(DataF) DATA X – Floating point input number
Returns
(DataF) RESULT – Return arc sinus geometry value

CMD: ACOS = 0x0F
Arguments
(DataF) DATA X – Floating point input number
Returns
(DataF) RESULT – Return arc cosine geometry value

CMD: ATAN = 0x10
Arguments
(DataF) DATA X – Floating point input number
Returns
(DataF) RESULT – Return arc tangent geometry value

CMD: MOD8 = 0x11
Arguments
(Data8) DATA X – Division numerator
(Data8) DATA Y – Division denominator
Returns
(Data8) RESULT – Result of modulo operation
Description
Modulo Data8, fmod(x,y)

CMD: MOD16 = 0x12
Arguments
(Data16) DATA X – Division numerator
(Data16) DATA Y – Division denominator
Returns
(Data16) RESULT – Result of modulo operation
Description
Modulo Data16, fmod(x,y)

CMD: MOD32 = 0x13

Arguments

(Data32) DATA X – Division numerator

(Data32) DATA Y – Division denominator

Returns

(Data32) RESULT – Result of modulo operation

Description

Modulo Data32, fmod(x,y)

CMD: POW = 0x14

Arguments

(DataF) DATA X – Value which is lifted to the power of Y

(DataF) DATA Y – Power numerator

Returns

(DataF) RESULT – Result of power operation

Description

Exponent, R = powf(x,y)

CMD: TRUNC = 0x15

Arguments

(DataF) DATA X – Value which are to be truncated

(Data8) DATA P – Precision [0-9]

Returns

(DataF) RESULT – Truncated value

Description

Truncate, R = (float)((int)(x*pow(y)))/ pow(Y)

4.3 Logic operations

Instruction	**opOr8 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x20
Arguments	(Data8) SOURCE1 – Source 1 value / index
	(Data8) SOURCE2 – Source 2 value / index
Result	(Data8) DESTINATION – Result value = Value 1 | Value 2
Dispatch status	Unchanged
Description	This function enables a program to or two 8 bit values together

Instruction	**opOr16 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x21
Arguments	(Data16) SOURCE1 – Source 1 value / index
	(Data16) SOURCE2 – Source 2 value / index
Result	(Data16) DESTINATION – Result value = Value 1 | Value 2
Dispatch status	Unchanged
Description	This function enables a program to or two 16 bit values together

Instruction	**opOr32 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x22
Arguments	(Data32) SOURCE1 – Source 1 value / index
	(Data32) SOURCE2 – Source 2 value / index
Result	(Data32) DESTINATION – Result value = Value 1 | Value 2
Dispatch status	Unchanged
Description	This function enables a program to or two 32 bit values together

Instruction	**opAnd8 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x24
Arguments	(Data8) SOURCE1 – Source 1 value / index
	(Data8) SOURCE2 – Source 2 value / index
Result	(Data8) DESTINATION – Result value = Value 1 & Value 2
Dispatch status	Unchanged
Description	This function enables a program to and two 8 bit values together

Instruction	**opAnd16 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x25
Arguments	(Data16) SOURCE1 – Source 1 value / index
	(Data16) SOURCE2 – Source 2 value / index
Result	(Data16) DESTINATION – Result value = Value 1 & Value 2
Dispatch status	Unchanged
Description	This function enables a program to and two 16 bit values together

Instruction	**opAnd32 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x26
Arguments	(Data32) SOURCE1 – Source 1 value / index (Data32) SOURCE2 – Source 2 value / index
Result	(Data32) DESTINATION – Result value = Value 1 & Value 2
Dispatch status	Unchanged
Description	This function enables a program to and two 32 bit values together

Instruction	**opXor8 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x28
Arguments	(Data8) SOURCE1 – Source 1 value / index (Data8) SOURCE2 – Source 2 value / index
Result	(Data8) DESTINATION – Result value = Value 1 ^ Value 2
Dispatch status	Unchanged
Description	This function enables a program to xor two 8 bit values together

Instruction	**opXor16 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x29
Arguments	(Data16) SOURCE1 – Source 1 value / index (Data16) SOURCE2 – Source 2 value / index
Result	(Data16) DESTINATION – Result value = Value 1 ^ Value 2
Dispatch status	Unchanged
Description	This function enables a program to xor two 16 bit values together

Instruction	**opXor32 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x2A
Arguments	(Data32) SOURCE1 – Source 1 value / index (Data32) SOURCE2 – Source 2 value / index
Result	(Data32) DESTINATION – Result value = Value 1 ^ Value 2
Dispatch status	Unchanged
Description	This function enables a program to xor two 32 bit values together

Instruction	**opRl8 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x2C
Arguments	(Data8) SOURCE1 – Source 1 value / index (Data8) SOURCE2 – Source 2 value / index
Result	(Data8) DESTINATION – Result value = Value 1 << Value 2
Dispatch status	Unchanged
Description	This function enables a program to rotate left an 8 bit value

Instruction	**opRl16 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x2D
Arguments	(Data16) SOURCE1 – Source 1 value / index (Data16) SOURCE2 – Source 2 value / index
Result	(Data16) DESTINATION – Result value = Value 1 << Value 2
Dispatch status	Unchanged
Description	This function enables a program to rotate left an 16 bit value

Instruction	**opRl32 (SOURCE1, SOURCE2, DESTINATION)**
Opcode	0x2E
Arguments	(Data32) SOURCE1 – Source 1 value / index (Data32) SOURCE2 – Source 2 value / index
Result	(Data32) DESTINATION – Result value = Value 1 << Value 2
Dispatch status	Unchanged
Description	This function enables a program to rotate left an 32 bit value

4.4 Memory move operations

Instruction	**opInit_Bytes (DESTINATION, LENGTH, SOURCE)**
Opcode	0x2F
Arguments	(Data32) LENGTH – Number of elements to initiate (Data8) SOURCE – First element to initiate data8 array with
Return	(Data8) DESTINATION – First element in data8 array to be initiated
Dispatch status	Unchanged
Description	This function enables moving a number of data8 from BYTE STREAM to memory DESTINATION START.

Instruction	**opMove8_8 (SOURCE, DESTINATION)**
Opcode	0x30
Arguments	(Data8) SOURCE – Source value / index
Return	(Data8) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 8 bit source to 8 bit destination

Instruction	**opMove8_16 (SOURCE, DESTINATION)**
Opcode	0x31
Arguments	(Data8) SOURCE – Source value / index
Return	(Data16) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 8 bit source to 16 bit destination

Instruction	**opMove8_32 (SOURCE, DESTINATION)**
Opcode	0x32
Arguments	(Data8) SOURCE – Source value / index
Return	(Data32) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 8 bit source to 32 bit destination

Instruction	**opMove8_F (SOURCE, DESTINATION)**
Opcode	0x33
Arguments	(Data8) Source – Source value / index
Return	(DataF) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 8 bit source to floating point destination

Instruction	**opMove16_8 (SOURCE, DESTINATION)**
Opcode	0x34
Arguments	(Data16) SOURCE – Source value / index
Return	(Data8) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 16 bit source to 8 bit destination

Instruction	**opMove16_16 (SOURCE, DESTINATION)**
Opcode	0x35
Arguments	(Data16) SOURCE – Source value / index
Return	(Data16) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 16 bit source to 16 bit destination

Instruction	**opMove16_32 (SOURCE, DESTINATION)**
Opcode	0x36
Arguments	(Data16) SOURCE – Source value / index
Return	(Data32) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 16 bit source to 32 bit destination

Instruction	**opMove16_F (SOURCE, DESTINATION)**
Opcode	0x37
Arguments	(Data16) SOURCE – Source value / index
Return	(DataF) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 16 bit source to floating point destination

Instruction	**opMove32_8 (SOURCE, DESTINATION)**
Opcode	0x38
Arguments	(Data32) SOURCE – Source value / index
Return	(Data8) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 32 bit source to 8 bit destination

Instruction	**opMove32_16 (SOURCE, DESTINATION)**
Opcode	0x39
Arguments	(Data32) SOURCE – Source value / index
Return	(Data16) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 32 bit source to 16 bit destination

Instruction	**opMove32_32 (SOURCE, DESTINATION)**
Opcode	0x3A
Arguments	(Data32) SOURCE – Source value / index
Return	(Data32) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 32 bit source to 32 bit destination

Instruction	**opMove32_F (SOURCE, DESTINATION)**
Opcode	0x3B
Arguments	(Data32) SOURCE – Source value / index
Return	(DataF) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from 32 bit source to floating point destination

Instruction	**opMoveF__8 (SOURCE, DESTINATION)**
Opcode	0x3C
Arguments	(DataF) SOURCE – Source value / index
Return	(Data8) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from floating point source to 8 bit destination

Instruction	**opMoveF__16 (SOURCE, DESTINATION)**
Opcode	0x3D
Arguments	(DataF) SOURCE – Source value / index
Return	(Data16) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from floating point source to 16 bit destination

Instruction	**opMoveF__32 (SOURCE, DESTINATION)**
Opcode	0x3E
Arguments	(DataF) SOURCE – Source value / index
Return	(Data32) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from floating point source to 32 bit destination

Instruction	**opMoveF__F (SOURCE, DESTINATION)**
Opcode	0x3F
Arguments	(DataF) SOURCE – Source value / index
Return	(DataF) DESTINATION – Destination value / index
Dispatch status	Unchanged
Description	Move value from floating point source to floating point destination

Instruction	**opRead8 (SOURCE, INDEX, DESTINATION)**
Opcode	0xC8
Arguments	(Data8) SOURCE – First value in array of values (Data8) INDEX – Index into array which are to be read
Return	(Data8) DESTINATION – Destination value / index to receive read data
Dispatch status	Unchanged
Description	Read 8 bit value from source array and store within destination

Instruction	**opRead16 (SOURCE, INDEX, DESTINATION)**
Opcode	0xC9
Arguments	(Data16) SOURCE – First value in array of values (Data8) INDEX – Index into array which are to be read
Return	(Data16) DESTINATION – Destination value / index to receive read data
Dispatch status	Unchanged
Description	Read 16 bit value from source array and store within destination

Instruction	**opRead32 (SOURCE, INDEX, DESTINATION)**
Opcode	0xCA
Arguments	(Data32) SOURCE – First value in array of values (Data8) INDEX – Index into array which are to be read
Return	(Data32) DESTINATION – Destination value / index to receive read data
Dispatch status	Unchanged
Description	Read 32 bit value from source array and store within destination

Instruction	**opReadF (SOURCE, INDEX, DESTINATION)**
Opcode	0xCB
Arguments	(DataF) SOURCE – First value in array of values (Data8) INDEX – Index into array which are to be read
Return	(DataF) DESTINATION – Destination value / index to receive read data
Dispatch status	Unchanged
Description	Read floating point value from source array and store within destination

Instruction	**opWrite8 (SOURCE, INDEX, DESTINATION)**
Opcode	0xCC
Arguments	(Data8) SOURCE – Variable to write (Data8) INDEX – Index into array were data are to be written (Data8) DESTINATION – Array to receive data
Dispatch status	Unchanged
Description	Write 8 bit value from source into destination array and index

Instruction	**opWrite16 (SOURCE, INDEX, DESTINATION)**
Opcode	0xCD
Arguments	(Data16) SOURCE – Variable to write (Data8) INDEX – Index into array were data are to be written (Data16) DESTINATION – Array to receive data
Dispatch status	Unchanged
Description	Write 16 bit value from source into destination array and index

Instruction	**opWrite32 (SOURCE, INDEX, DESTINATION)**
Opcode	0xCE
Arguments	(Data32) SOURCE – Variable to write (Data8) INDEX – Index into array were data are to be written (Data32) DESTINATION – Array to receive data
Dispatch status	Unchanged
Description	Write 32 bit value from source into destination array and index

Instruction	**opWriteF (SOURCE, INDEX, DESTINATION)**
Opcode	0xCF
Arguments	(DataF) SOURCE – Variable to write (Data8) INDEX – Index into array were data are to be written (DataF) DESTINATION – Array to receive data
Dispatch status	Unchanged
Description	Write floating point value from source into destination array and index

4.5 Program branch operations

Instruction	**opJr (OFFSET)**
Opcode	0x40
Arguments	(Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch unconditionally relative to current position

Instruction	**opJr__False (FLAG, OFFSET)**
Opcode	0x41
Arguments	(Data8) FLAG (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative given that flag is FALSE (Zero)

Instruction	**opJr__True (OFFSET)**
Opcode	0x42
Arguments	(Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative given that flag is TRUE (None zero)

Instruction	**opJr__Nan (VALUE, OFFSET)**
Opcode	0x43
Arguments	(DataF) VALUE (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if value is NAN (Not a number)

Instruction	**opJr__Lt8 (LEFT, RIGHT, OFFSET)**
Opcode	0x64
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is less than RIGHT value

Instruction	**opJr__Lt16 (LEFT, RIGHT, OFFSET)**
Opcode	0x65
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is less than RIGHT value

Instruction	**opJr__Lt32 (LEFT, RIGHT, OFFSET)**
Opcode	0x66
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is less than RIGHT value

Instruction	**opJr__LtF (LEFT, RIGHT, OFFSET)**
Opcode	0x67
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is less than RIGHT value

Instruction	**opJr__Gt8 (LEFT, RIGHT, OFFSET)**
Opcode	0x68
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is greater than RIGHT value

Instruction	**opJr__Gt16 (LEFT, RIGHT, OFFSET)**
Opcode	0x69
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is greater than RIGHT value

Instruction	**opJr__Gt32 (LEFT, RIGHT, OFFSET)**
Opcode	0x6A
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is greater than RIGHT value

Instruction	**opJr__GtF (LEFT, RIGHT, OFFSET)**
Opcode	0x6B
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is greater than RIGHT value

Instruction	**opJr__Eq8 (LEFT, RIGHT, OFFSET)**
Opcode	0x6C
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is equal to RIGHT value

Instruction	**opJr__Eq16 (LEFT, RIGHT, OFFSET)**
Opcode	0x6D
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is equal to RIGHT value

Instruction	**opJr__Eq32 (LEFT, RIGHT, OFFSET)**
Opcode	0x6E
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is equal to RIGHT value

Instruction	**opJr__EqF (LEFT, RIGHT, OFFSET)**
Opcode	0x6F
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is equal to RIGHT value

Instruction	**opJr__Neq8 (LEFT, RIGHT, OFFSET)**
Opcode	0x70
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is not equal to RIGHT value

Instruction	**opJr__Neq16 (LEFT, RIGHT, OFFSET)**
Opcode	0x71
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is not equal to RIGHT value

Instruction	**opJr__Neq32 (LEFT, RIGHT, OFFSET)**
Opcode	0x72
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is not equal to RIGHT value

Instruction	**opJr__NeqF (LEFT, RIGHT, OFFSET)**
Opcode	0x73
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is not equal to RIGHT value

Instruction	**opJr__Lteq8 (LEFT, RIGHT, OFFSET)**
Opcode	0x74
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is less than or equal to RIGHT value

Instruction	**opJr__Lteq16 (LEFT, RIGHT, OFFSET)**
Opcode	0x75
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is less than or equal to RIGHT value

Instruction	**opJr__Lteq32 (LEFT, RIGHT, OFFSET)**
Opcode	0x76
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is less than or equal to RIGHT value

Instruction	**opJr__LteqF (LEFT, RIGHT, OFFSET)**
Opcode	0x77
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is less than or equal to RIGHT value

Instruction	**opJr__Gteq8 (LEFT, RIGHT, OFFSET)**
Opcode	0x78
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is greater than or equal to RIGHT value

Instruction	**opJr__Gteq16 (LEFT, RIGHT, OFFSET)**
Opcode	0x79
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is greater than or equal to RIGHT value

Instruction	**opJr__Gteq32 (LEFT, RIGHT, OFFSET)**
Opcode	0x7A
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is greater than or equal to RIGHT value

Instruction	**opJr__GteqF (LEFT, RIGHT, OFFSET)**
Opcode	0x7B
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic (Data32) OFFSET – Number of bytes to branch
Dispatch status	Unchanged
Description	Program branch relative if LEFT value is greater than or equal to RIGHT value

4.6 Comparison operation

Instruction	**opCp_Lt8 (LEFT, RIGHT, FLAG)**
Opcode	0x44
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic
Result	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is less than RIGHT and set flag accordingly

Instruction	**opCp_Lt16 (LEFT, RIGHT, FLAG)**
Opcode	0x45
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic
Result	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is less than RIGHT and set flag accordingly

Instruction	**opCp_Lt32 (LEFT, RIGHT, FLAG)**
Opcode	0x46
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic
Result	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is less than RIGHT and set flag accordingly

Instruction	**opCp_LtF (LEFT, RIGHT, FLAG)**
Opcode	0x47
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic
Result	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is less than RIGHT and set flag accordingly

Instruction	**opCp_Gt8 (LEFT, RIGHT, FLAG)**
Opcode	0x48
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is greater than RIGHT and set flag accordingly

Instruction	**opCp_Gt16 (LEFT, RIGHT, FLAG)**
Opcode	0x49
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is greater than RIGHT and set flag accordingly

Instruction	**opCp_Gt32 (LEFT, RIGHT, FLAG)**
Opcode	0x4A
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is greater than RIGHT and set flag accordingly

Instruction	**opCp_GtF (LEFT, RIGHT, FLAG)**
Opcode	0x4B
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is greater than RIGHT and set flag accordingly

Instruction	**opCp_Eq8 (LEFT, RIGHT, FLAG)**
Opcode	0x4C
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is equal to RIGHT and set flag accordingly

Instruction	**opCp_Eq16 (LEFT, RIGHT, FLAG)**
Opcode	0x4D
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is equal to RIGHT and set flag accordingly

Instruction	**opCp_Eq32 (LEFT, RIGHT, FLAG)**
Opcode	0x4E
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is equal to RIGHT and set flag accordingly

Instruction	**opCp_EqF (LEFT, RIGHT, FLAG)**
Opcode	0x4F
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is equal to RIGHT and set flag accordingly

Instruction	**opCp_Neq8 (LEFT, RIGHT, FLAG)**
Opcode	0x50
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is not equal to RIGHT and set flag accordingly

Instruction	**opCp_Neq16 (LEFT, RIGHT, FLAG)**
Opcode	0x51
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is not equal to RIGHT and set flag accordingly

Instruction	**opCp_Neq32 (LEFT, RIGHT, FLAG)**
Opcode	0x52
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is not equal to RIGHT and set flag accordingly

Instruction	**opCp_NeqF (LEFT, RIGHT, FLAG)**
Opcode	0x53
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is not equal to RIGHT and set flag accordingly

Instruction	**opCp_Lteq8 (LEFT, RIGHT, FLAG)**
Opcode	0x54
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is less than or equal to RIGHT and set flag accordingly

Instruction	**opCp_Lteq16 (LEFT, RIGHT, FLAG)**
Opcode	0x55
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is less than or equal to RIGHT and set flag accordingly

Instruction	**opCp_Lteq32 (LEFT, RIGHT, FLAG)**
Opcode	0x56
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is less than or equal to RIGHT and set flag accordingly

Instruction	**opCp_LteqF (LEFT, RIGHT, FLAG)**
Opcode	0x57
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is less than or equal to RIGHT and set flag accordingly

Instruction	**opCp_Gteq8 (LEFT, RIGHT, FLAG)**
Opcode	0x58
Arguments	(Data8) LEFT – Left value within comparison logic (Data8) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is greater than or equal to RIGHT and set flag accordingly

Instruction	**opCp_Gteq16 (LEFT, RIGHT, FLAG)**
Opcode	0x59
Arguments	(Data16) LEFT – Left value within comparison logic (Data16) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is greater than or equal to RIGHT and set flag accordingly

Instruction	**opCp_Gteq32 (LEFT, RIGHT, FLAG)**
Opcode	0x5A
Arguments	(Data32) LEFT – Left value within comparison logic (Data32) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is greater than or equal to RIGHT and set flag accordingly

Instruction	**opCp_GteqF (LEFT, RIGHT, FLAG)**
Opcode	0x5B
Arguments	(DataF) LEFT – Left value within comparison logic (DataF) RIGHT – Right value within comparison logic
Return	(Data8) FLAG – Result of comparison (None zero if true)
Dispatch status	Unchanged
Description	Enables a program to identify if LEFT is greater than or equal to RIGHT and set flag accordingly

4.7 Program select operations

Instruction	**opSelect8 (FLAG, SOURCE1, SOURCE2, *RESULT)**
Opcode	0x5C
Arguments	(Data8) FLAG – Function as determined for operation (Data8) SOURCE1 – Source 1 value / index (Data8) SOURCE2 – Source 2 value / index
Return	(Data8) *RESULT – Destination for value
Dispatch status	Unchanged
Description	Enables a program to move value based on flag. If FLAG is set source 1 will be moved to result. If FLAG is not set, source 2 will be moved to result.

Instruction	**opSelect16 (FLAG, SOURCE1, SOURCE2, *RESULT)**
Opcode	0x5D
Arguments	(Data8) FLAG – Function as determined for operation (Data16) SOURCE1 – Source 1 value / index (Data16) SOURCE2 – Source 2 value / index
Return	(Data16) *RESULT – Destination for value
Dispatch status	Unchanged
Description	Enables a program to move value based on flag. If FLAG is set source 1 will be moved to result. If FLAG is not set, source 2 will be moved to result.

Instruction	**opSelect32 (FLAG, SOURCE1, SOURCE2, *RESULT)**
Opcode	0x5E
Arguments	(Data8) FLAG – Function as determined for operation (Data32) SOURCE1 – Source 1 value / index (Data32) SOURCE2 – Source 2 value / index
Return	(Data32) *RESULT – Destination for value
Dispatch status	Unchanged
Description	Enables a program to move value based on flag. If FLAG is set source 1 will be moved to result. If FLAG is not set, source 2 will be moved to result.

Instruction	**opSelectF (FLAG, SOURCE1, SOURCE2, *RESULT)**
Opcode	0x5F
Arguments	(Data8) FLAG – Function as determined for operation (DataF) SOURCE1 – Source 1 value / index (DataF) SOURCE2 – Source 2 value / index
Return	(DataF) *RESULT – Destination for value
Dispatch status	Unchanged
Description	Enables a program to move value based on flag. If FLAG is set source 1 will be moved to result. If FLAG is not set, source 2 will be moved to result.

4.8 Input port operations

Instruction	**opInput_Device_List (LENGTH, ARRAY, CHANGED)**
Opcode	0x98
Arguments	(Data8) LENGTH – Maximum number of devices types (Normally 32)
Return	(Data8) ARRAY – First element of data8 array of types (Normally 32)
	(Data8) CHANGED – Changed status
Dispatch status	Unchanged
Description	Enables a program to read all available devices on input chain

Instruction	**opInput_Device (CMD, …)**
Opcode	0x99
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	Read information about external sensor device

CMD: GET_FORMAT = 0x02

Arguments

(Data8) LAYER – Specify chain layer number [0-3]

(Data8) NO – Port number

Returns

(Data8) Datasets – Number of data sets

(Data8) FORMAT – Format [0-3], (0: 8-bit, 1: 16-bit, 2: 32-bit, 3: Float point)

(Data8) MODES – Number of modes [1-8]

(Data8) VIEW – Number of modes visible within port view app [1-8]

CMD: CAL_MINMAX = 0x03

Arguments

(Data8) TYPE – Device type (Please reference section 0)

(Data8) MODE – Device mode [0-7]

(Data32) CAL_MIN – 32 bit raw minimum value (Zero point)

(Data32) CAL_MAX – 32 bit raw maximum value (Full scale)

Description

Apply new minimum and maximum raw value for device type to be used in scaling PCT and SI value

CMD: CAL_DEFAULT = 0x04

Arguments

(Data8) TYPE – Device type (Please reference section 0)

(Data8) MODE – Device mode [0-7]

Description

Apply the default minimum and maximum raw value for device type to be used in scaling PCT and SI value

CMD: GET__TYPEMODE = 0x05
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
Returns
(Data8) TYPE – See device type list (Please reference section 0)
(Data8) MODE – Device mode [0-7]

CMD: GET__SYMBOL = 0x06
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
(Data8) LENGTH – Maximum length of string returned (-1 = No check)
Returns
(Data8) DESTINATION – String variable or handle to string

CMD: CAL__MIN = 0x07
Arguments
(Data8) TYPE – Device type (Please reference section 0)
(Data8) MODE – Device mode [0-7]
(Data32) CAL__MIN – 32 bit raw minimum value (Zero point)
Description
Apply new minimum raw value for device type to be used in scaling PCT and SI value

CMD: CAL__MAX = 0x08
Arguments
(Data8) TYPE – Device type (Please reference section 0)
(Data8) MODE – Device mode [0-7]
(Data32) CAL__MAX – 32 bit raw maximum value (Full scale)
Description
Apply new minimum raw value for device type to be used in scaling PCT and SI value

CMD: SETUP = 0x09
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
(Data8) REPEAT – Repeat setup/read “Repeat” times (0 = Infinite)
(Data16) TIME – Time between repeats [10-1000 mS] (0 = 10)
(Data8) WRLNG – No of bytes to write
(Data8) WRDATA – Array handle (Data8) of data to write
(Data8) RDLNG – No of bytes to read
Returns
(Data8) RDDATA – Array handle (Data8) for data to read
Description
Generic setup / read for IIC sensor

CMD: CLR_ALL = 0x0A
Arguments
(Data8) LAYER – Specify chain layer number [0-3] (-1 = All)
Description
Clear all device counters and values

CMD: GET_RAW = 0x0B
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
Returns
(Data32) Value – 32 bit raw value on the given sensor port

CMD: GET_CONNECTION = 0x0C
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
Returns
(Data8) CONN – Connection type, see documentation below

Connection type:

0x6F : CONN_UNKNOW	: Fake
0x75 : CONN_DAISYCHAIN	: Daisy chained
0x76 : CONN_NXT_COLOR	: NXT Color sensor
0x77 : CONN_NXT_DUMB	: NXT analog sensor
0x78 : CONN_NXT_IIC	: NXT IIC sensor
0x79 : CONN_INPUT_DUMB	: EV3 input device with ID resistor
0x7A : CONN_INPUT_UART	: EV3 input UART sensor
0x7B: CONN_OUTPUT_DUMB	: EV3 output device with ID resistor
0x7C: CONN_OUTPUT_INTELLIGENT	: EV3 output device with communication
0x7D: CONN_OUTPUT_TACHO	: EV3 Tacho motor with ID resistor
0x7E: CONN_NONE	: Port empty or not available
0x7F: CONN_ERROR	: Port not empty and type is invalid

CMD: STOP_ALL= 0x0D (Stop all devices)
Arguments
(Data8) LAYER – Specify chain layer number [0-3] (-1 = All)

CMD: GET_NAME = 0x15
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
(Data8) LENGTH – Maximum length of string returned (-1 = No check)
Returns
(Data8) DESTINATION – String variable or handle to string

CMD: GET__MODENAME = 0x16
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
(Data8) MODE – Mode number
(Data8) LENGTH – Maximum length of string returned (-1 = No check)
Returns
(Data8) DESTINATION – String variable or handle to string

CMD: GET__FIGURES = 0x18
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
Returns
(Data8) FIGURES – Total number of figures (Inclusive decimal points and decimals)
(Data8) DECIMALES – Number of decimals

CMD: GET__CHANGES = 0x19
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
Returns
(DataF) VALUE1 – Positive changes since last clear. (Button pressed)

CMD: CLR__CHANGES = 0x1A
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
Description
Clear changes and bumps

CMD: READY__PCT = 0x1B
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
(Data8) TYPE – Specify device type (0 = Don't change type)
(Data8) MODE – Device mode [0-7] (-1 = Don't change mode)
(Data8) VALUES – Number of return values
Returns (Depending on number of data samples requested in (VALUES))
(Data8) VALUE1 – First value received from sensor in the specified mode

CMD: READY_RAW = 0x1C
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
(Data8) TYPE – Specify device type (0 = Don't change type)
(Data8) MODE – Device mode [0-7] (-1 = Don't change mode)
(Data8) VALUES – Number of return values
Returns (Depending on number of data samples requested in (VALUES))
(Data32) VALUE1 – First value received from sensor in the specified mode

CMD: READY_SI = 0x1D
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
(Data8) TYPE – Specify device type (0 = Don't change type)
(Data8) MODE – Device mode [0-7] (-1 = Don't change mode)
(Data8) VALUES – Number of return values
Returns (Depending on number of data samples requested in (VALUES))
(DataF) VALUE1 – First value received from sensor in the specified mode

CMD: GET_MINMAX = 0x1E
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
Returns
(DataF) MIN – Minimum SI value
(DataF) MAX – Maximum SI value

CMD: GET_BUMPS = 0x1F
Arguments
(Data8) LAYER – Specify chain layer number [0-3]
(Data8) NO – Port number
Returns
(DataF) VALUE1 – Negative changes since last clear. (Button release)

Instruction	**opInput_Read (LAYER, NO, TYPE, MODE, PCT)**
Opcode	0x9A
Arguments	(Data8) LAYER – Specify chain layer number [0-3] (Data8) NO – Port number (Data8) TYPE – Specify device type (0 = Don't change type) (Data8) MODE – Device mode [0-7] (-1 = Don't change mode)
Return	(Data8) PCT – Percentage value from device
Dispatch status	Unchanged
Description	This function enables reading specific device and mode in percentage

Instruction	**opInput_Test (LAYER, NO, BUSY)**
Opcode	0x9B
Arguments	(Data8) LAYER – Specify chain layer number [0-3]
	(Data8) NO – Port number
Return	(Data8) BUSY – Device busy flag (0 = Ready, 1 = Busy)
Dispatch status	Unchanged
Description	This function enables testing if a sensor is busy changing type or mode

Instruction	**opInput_Ready (LAYER, NO)**
Opcode	0x9C
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3]
	(Data8) NO – Port number
Dispatch status	Can change to BUSYBREAK
Description	This function enables a program to wait for a device to be ready (Wait for valid data)

Instruction	**opInput_ReadSI (LAYER, NO, TYPE, MODE, SI)**
Opcode	0x9D
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3]
	(Data8) NO – Port number
	(Data8) TYPE – Specify device type (0 = Don't change type)
	(Data8) MODE – Device mode [0 - 7] (-1 = Don't change mode)
Return	(DataF) SI – SI unit value from device
Dispatch status	Unchanged
Description	This function enables reading specific device and mode in SI units

Instruction	**opInput_ReadExt (LAYER, NO, TYPE, MODE, FORMAT, VALUES, VALUE1)**
Opcode	0x9E
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3]
	(Data8) NO – Port number
	(Data8) TYPE – Specify device type (0 = Don't change type)
	(Data8) MODE – Device mode [0 - 7] (-1 = Don't change mode)
	(Data8) FORMAT – Format selection (PCT, RAW, SI,…)
	(Data8) Values – Number of return values
Return (Depending on number of data samples requested in (VALUES))	
	(FORMAT) VALUE1 – First value received from device in specified mode
Dispatch status	Unchanged
Description	This function enables reading multiple data from external device simultanously

Instruction	**opInput_Write (LAYER, NO, BYTES, DATA)**
Opcode	0x9F
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3]
	(Data8) NO – Port number
	(Data8) BYTES – No of bytes to write [1 - 32]
	(Data8) DATA – First byte of data8 array to write
Dispatch status	Can change to BUSYBREAK
Description	This function enables writting data to extenal digital devices

4.9 Output port operations

Instruction	**opOutput_Set_Type (LAYER, NO, TYPE)**
Opcode	0xA1
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NO – Port number [0 - 3] (Data8) TYPE – Output device type, (0x07: Large motor, Medium motor = 0x08)
Dispatch status	Unchanged
Description	This function enables specifying the output device type

Instruction	**opOutput_Reset (LAYER, NOS)**
Opcode	0xA2
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F]
Dispatch status	Unchanged
Description	This function enables restting the tacho count for the individual output ports

Instruction	**opOutput_Stop (LAYER, NOS, BRAKE)**
Opcode	0xA3
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F] (Data8) BRAKE – Specify break level [0: Float, 1: Break]
Dispatch status	Unchanged
Description	This function enables restting the tacho count for the individual output ports

Instruction	**opOutput_Power (LAYER, NOS, POWER)**
Opcode	0xA4
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F] (Data8) POWER – Specify output speed [-100 – 100 %]
Dispatch status	Unchanged
Description	This function enables setting the output percentage power on the output ports

Instruction	**opOutput_Speed (LAYER, NOS, SPEED)**
Opcode	0xA5
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F] (Data8) SPEED – Specify output speed [-100 – 100 %]
Dispatch status	Unchanged
Description	This function enables setting the output percentage speed on the output ports. This modes automatically enables speed control, which means the system will automatically adjust the power to keep the specified speed.
Instruction	**opOutput_Start (LAYER, NOS)**
Opcode	0xA6
Arguments	(Data8) LAYER – Specify chain layer number, [0 - 3] (Data8) NOS – Output bit field, [0x00 – 0x0F]
Dispatch status	Unchanged
Description	This function enables starting the specified output port.

Instruction	**opOutput_Polarity (LAYER, NOS, POL)**
Opcode	0xA7
Arguments	(Data8) LAYER – Specify chain layer number, [0 - 3] (Data8) NOS – Output bit field, [0x00 – 0x0F] (Data8) POL – Polarity [-1, 0, 1], see documentation below
Dispatch status	Unchanged
Description	This function enables starting the specified output port. Polarity parameter: -1 : Motor will run backward 0 : Motor will run opposite direction 1 : Motor will run forward

Instruction	**opOutput_Read (LAYER, NO, *SPEED, *TACHO)**
Opcode	0xA8
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NO – Port number [0 - 3] (Data8) *SPEED – Output speed level detected, [-100 - 100] (Data32) *TACHO – Current output tacho count
Dispatch status	Unchanged
Description	This function enables reading current motor speed and tacho count level.

Instruction	**opOutput_Test (LAYER, NOS, BUSY)**
Opcode	0xA9
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F]
Return	(Data8) BUSY – Output busy flag, [0 = Ready, 1 = Busy]
Dispatch status	Unchanged
Description	This function enables the program to test if a output port is busy.

Instruction	**opOutput_Ready (LAYER, NOS)**
Opcode	0xAA
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F]
Dispatch status	Can changed to BUSYBREAK
Description	Enables program execution to wait for output ready. (Wait for completion)

Instruction	**opOutput_Step_Power (LAYER, NOS, POWER, STEP1, STEP2, STEP3, BRAKE)**
Opcode	0xAC
Arguments	(Data8) LAYER – Specify chain layer number, [0 - 3] (Data8) NOS – Output bit field, [0x00 – 0x0F] (Data8) POWER – Power level, [-100 - 100] (Data32) STEP1 – Tacho pulses during ramp up (Data32) STEP2 – Tacho pulses during continues run (Data32) STEP3 – Tacho pulses during ramp down (Data8) BRAKE - Specify break level, [0: Float, 1: Break]
Dispatch status	Unchanged
Description	This function enables specifying a full motor power cycle in tacho counts. Step1 specifyes the power ramp up periode in tacho count, Step2 specifyes the constant power period in tacho counts, Step 3 specifyes the power down period in tacho counts.

Instruction	**opOutput_Time_Power (LAYER, NOS, POWER, STEP1, STEP2, STEP3, BRAKE)**
Opcode	0xAD
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F] (Data8) POWER – Power level, [-100 – 100] (Data32) STEP1 – Time in milliseconds for ramp up (Data32) STEP2 – Time in milliseconds for continues run (Data32) STEP3 – Time in milliseconds for ramp down (Data8) BRAKE - Specify break level [0: Float, 1: Break]
Dispatch status	Unchanged
Description	This function enables specifying a full motor power cycle in time. Step1 specifyes the power ramp up periode in milliseconds, Step2 specifyes the constant power period in milliseconds, Step 3 specifyes the power down period in milliseconds.

Instruction	**opOutput_Step_Speed (LAYER, NOS, SPEED, STEP1, STEP2, STEP3, BRAKE)**
Opcode	0xAE
Arguments	(Data8) LAYER – Specify chain layer number, [0 - 3] (Data8) NOS – Output bit field, [0x00 – 0x0F] (Data8) SPEED – Power level, [-100 - 100] (Data32) STEP1 – Tacho pulses during ramp up (Data32) STEP2 – Tacho pulses during continues run (Data32) STEP3 – Tacho pulses during ramp down (Data8) BRAKE - Specify break level, [0: Float, 1: Break]
Dispatch status	Unchanged
Description	This function enables specifying a full motor power cycle in tacho counts. The system will automatically adjust the power level to the motor to keep the specified output speed. Step1 specifyes the power ramp up periode in tacho count, Step2 specifyes the constant power period in tacho counts, Step 3 specifyes the power down period in tacho counts.

Instruction	**opOutput_Time_Speed (LAYER, NOS, SPEED, STEP1, STEP2, STEP3, BRAKE)**
Opcode	0xAF
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F] (Data8) SPEED – Power level, [-100 – 100] (Data32) STEP1 – Time in milliseconds for ramp up (Data32) STEP2 – Time in milliseconds for continues run (Data32) STEP3 – Time in milliseconds for ramp down (Data8) BRAKE - Specify break level [0: Float, 1: Break]
Dispatch status	Unchanged
Description	This function enables specifying a full motor power cycle in time. The system will automatically adjust the power level to the motor to keep the specified output speed. Step1 specifyes the power ramp up periode in milliseconds, Step2 specifyes the constant power period in milliseconds, Step 3 specifyes the power down period in milliseconds.

Instruction **opOutput_Step_Sync (LAYER, NOS, SPEED, TURN, STEP, BRAKE)**

Opcode	0xB0
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F] (Data8) SPEED – Power level, [-100 – 100] (Data16) TURN – Turn ratio, [-200 - 200], see documentation below (Data32) STEP – Tacho pulses, 0 = Infinite (Data8) BRAKE - Specify break level [0: Float, 1: Break]
Dispatch status	Unchanged
Description	This function enables synchonizing two motors. Synchonization should be used when motors should run as synchrone as possible, for example to archieve a model driving straight. Duration is specified in tacho counts. Turn ratio: 0 : Motor will run with same power 100 : One motor will run with specified power while the other will be close to zero 200: One motor will run with specified power forward while the other will run in the opposite direction at the same power level.

Instruction	**opOutput_Time_Sync (LAYER, NOS, SPEED, TURN, STEP, BRAKE)**
Opcode	0xB1
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F] (Data8) SPEED – Power level, [-100 – 100] (Data16) TURN – Turn ratio, [-200 - 200], see documentation below (Data32) TIME – Time in milliseconds, 0 = Infinite (Data8) BRAKE - Specify break level [0: Float, 1: Break]
Dispatch status	Unchanged
Description	This function enables synchonizing two motors. Synchonization should be used when motors should run as synchrone as possible, for example to archieve a model driving straight. Duration is specified in time.

Instruction	**opOutput_Clr_Count (LAYER, NOS)**
Opcode	0xB2
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F]
Dispatch status	Unchanged
Description	This function enables the program to clear the tacho count used as sensor input.

Instruction	**opOutput_Get_Count (LAYER, NOS, *TACHO)**
Opcode	0xB3
Arguments	(Data8) LAYER – Specify chain layer number [0 - 3] (Data8) NOS – Output bit field [0x00 – 0x0F] (Data32) *TACHO – Tacho count as sensor value
Dispatch status	Unchanged
Description	This function enables the program to read the tacho count as sensor input.

Instruction	**opOutput_Prg_Stop**
Opcode	0xB4
Arguments	
Dispatch status	Unchanged
Description	This function should be called a program end. It enables breaking the motor for a short period and right after floating the motors. The function relates to the layer on which it is executed.

4.10 Sound operations

Instruction	**opSound (CMD, ...)**
Opcode	0x94
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	Sound control entry

CMD: BREAK = 0x00 (Stop current sound playback)

CMD: TONE = 0x01
Arguments
(Data8) VOLUME – Specify volume for playback, [0 - 100]
(Data16) FREQUENCY – Specify frequency, [250 - 10000]
(Data16) DURATION – Specify duration in millisecond

CMD: PLAY = 0x02
Arguments
(Data8) VOLUME – Specify volume for playback, [0 - 100]
(Data8) NAME – First character in filename (Character string)

CMD: REPEAT = 0x03
Arguments
(Data8) VOLUME – Specify volume for playback, [0 - 100]
(Data8) NAME – First character in filename (Character string)

Instruction	**opSound_Test (BUSY)**
Opcode	0x95
Arguments	
Return	(Data8) BUSY – Output busy flag, [0 = Ready, 1 = Busy]
Dispatch status	Unchanged
Description	This function enables the program to test if sound is busy (Playing sound or tone)

Instruction	**opSound_Ready ()**
Opcode	0x96
Arguments	
Dispatch status	Can changed to BUSYBREAK
Description	Enables program execution to wait for sound ready. (Wait for completion)

4.11 Timer operations

Instruction	**opTimer_Wait (TIME, TIMER)**
Opcode	0x85
Arguments	(Data32) TIME – Time to wait in milliseonds (Data32) TIMER – Variable used for timing
Dispatch status	Unchanged
Description	This function enables setting up a time to wait in mS

Instruction	**opTimer_Ready (TIMER)**
Opcode	0x86
Arguments	(Data32) TIMER – Variable used for timing
Dispatch status	Can changed to BUSYBREAK
Description	This function enables program execution to wait until timer is ready / finished.

Instruction	**opTimer_Read (TIME)**
Opcode	0x87
Arguments	(Data32) TIME – Current time stamp from millisecond timer
Dispatch status	Unchanged
Description	This function enables program to read free running millisecond timer

Instruction	**opTimer_Read_Us (TIME)**
Opcode	0x8F
Arguments	(Data32) TIME – Current time stamp from microsecond timer
Dispatch status	Unchanged
Description	This function enables program to read free running microsecond timer

4.13 Communication operations

Instruction	**opCom_Ready (HARDWARE, *pNAME)**
Opcode	0xD0
Arguments	(Data8) HARDWARE, See further documentation below. (Data8) *pNAME - Name of the remote/own device (0 = Own adapter status)
Dispatch status	Can changed to BUSYBREAK
Description	This function enables the program to verify and wait if communication is busy. The hardware parameter can take the following values: 1: USB communication (Client interface) 2: Bluetooth communication interface 3: WiFi communication interface

Instruction	**opCom_Test (HARDWARE, *pNAME, BUSY)**
Opcode	0xD1
Arguments	(Data8) HARDWARE, See further documentation below. (Data8) *pNAME - Name of the remote/own device (0 = Own adapter status)
Return	(Data8) BUSY – Busy flag, [0: Ready, 1: Busy]
Dispatch status	Set to NOBREAK
Description	This function enables the program to test if communication is busy. The hardware parameter can take the following values: 1: USB communication (Client interface) 2: Bluetooth communication interface 3: WiFi communication interface

Instruction	**opCom_Read (CMD, ...)**
Opcode	0x91
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	Communication read entry ***CMD:*** COMMAND = 0x0E Arguments (Data32) LENGTH – Maximal code stream length Return (Data32) *IMAGE – Address of image (Program snippet/pointer to program) (Data32) *GLOBAL – Address of global variables (Mem for potential reply) (Data8) FLAG – Indicate if image is ready Description Read communication data

Instruction	**opCom_Write (CMD, ...)**
Opcode	0x92
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	Communication write entry

CMD: REPLY = 0x0E
Arguments
Return (Data32) *IMAGE – Address of image (Program snippet/pointer to program)
(Data32) *GLOBAL – Address of global variables
Description
Reply communication on incoming command

Instruction	**opMailbox_Open (NO, BOXNAME, TYPE, FIFOSIZE, VALUES)**
Opcode	0xD8
Arguments	(Data8) NO – Reference ID for mailbox. Maximum number of mailboxes is 30. (Data8) BOXNAME – Zero terminated string with mailbox name (Data8) TYPE – Data type used within mailbox, see details below. (Data8) FIFOSIZE – Not used at this point (Data8) VALUES – Number of data entries (data specified by type above)
Dispatch status	Can return FAILBREAK
Description	This function enables opening a mailbox to be used within the running program. Maximum mailbox size is 250 bytes. I.e. if type is string (DataS) then thee can only be 1 string of maximum 250 bytes (Incl. zero termination), or of array (DataA), then array size cannot be larger than 250 bytes.

Type data supported:
0x00 : Data8 8-bit integer value
0x01 : Data16 16-bit integer value
0x02 : Data32 32-bit integer value
0x03 : DataF Floating point data
0x04 : DataS Zero terminated string
0x05 : DataA Array handle
0x06 : DataV Variable type

Instruction	**opMailbox_Write (BRICKNAME, HARDWARE, BOXNAME, TYPE, VALUES, VALUE)**
Opcode	0xD9
Arguments	(Data8) BRICKNAME - Zero terminated string name of the receiving brick (Data8) HARDWARE – Transportation media (Data8) BOXNAME – Zero terminated string name of the receiving mailbox (Data8) TYPE – Data type, see details at opMailbox_Open command (Data8) VALUES – Number of values of the specified type (Depending on TYPE) VALUE – Actual data to be send
Dispatch status	Can return FAILBREAK
Description	This function enables a running program to write data to another bricks mailbox. When using data type DataS, then a zero terminated string is expected. If data type is DataA, then number of bytes to be send is equal to array size. If brickname is left empty, then all connected devices will receive the mailbox message.

Instruction	**opMailbox_Read (NO, LENGTH, VALUES, VALUE)**
Opcode	0xDA
Arguments	(Data8) NO – Message box ID from where the message is to be read (Data8) LENGTH – Maximum bytes to be read (Data8) VALUES – Number of value to read
Return	(Depending on mailbox settings) VALUE – Data from the message box
Dispatch status	Can return FAILBREAK
Description	This function enables a running program read data from a specific mailbox.

Instruction	**opMailbox_Test (NO, BUSY)**
Opcode	0xDB
Arguments	(Data8) NO – Message box ID
Return	(Data8) BUSY – Busy equal TRUE if no new message have been received
Dispatch status	Can return FAILBREAK
Description	This function enables a running program to test if a new message have been received.

Instruction	**opMailbox_Ready (NO)**
Opcode	0xDC
Arguments	(Data8) NO – Message box ID
Dispatch status	Can return FAILBREAK
Description	This function enables a running program to wait until a message have been received within the specified mail box.

Instruction	**opMailbox_Close (NO)**
Opcode	0xDD
Arguments	(Data8) NO – Message box ID
Dispatch status	Can return FAILBREAK
Description	This function enables a running program to close an open mail box.

Instruction	**opCom_Get (CMD, ...)**

Opcode	0xD3
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Can return FAILBREAK
Description	Communication get entry

CMD: GET_ON_OFF = 0x01
Arguments
(Data8) HARDWARE – See command opCom_Test for details
Return
(Data8) ACTIVE – Visible [0, 1]

CMD: GET_VISIBLE = 0x02
Arguments
(Data8) HARDWARE – See command opCom_Test for details
Return
(Data8) Visible – Visible [0, 1]

CMD: GET_RESULT = 0x04
Arguments
(Data8) HARDWARE – See command opCom_Test for details
(Data8) ITEM – Name index
Return
(Data8) Result – Status [0: Ok, 1: Busy, 2: Fail, 4: Stop]
Description
Get result of the command that is being executed. This could be a search or a connection request.

CMD: GET_PIN = 0x05
Arguments
(Data8) HARDWARE – See command opCom_Test for details
(Data8) NAME – First character in string
(Data8) LENGTH – Max length of returned string
Return
(Data8) PINCODE – Actual pin code

CMD: SEARCH_ITEMS = 0x08
Arguments
(Data8) HARDWARE – See command opCom_Test for details
Return
(Data8) ITEMS – Number of items in search list
Description
Get number of item from search. After a search has been completed, SEARCH ITEMS will return the number of remote devices found.

CMD: SEARCH_ITEM = 0x09

Arguments

(Data8) HARDWARE – See command opCom__Test for details
(Data8) ITEM - Item index within the search list
(Data8) LENGTH – Max length of returned string

Return

(Data8) NAME – First character in character string (Zero terminated)
(Data8) PAIRED – Indicate if paired [0: Not paired, 1: Paired]
(Data8) Connected – Indicate if connected [0: Not connected, 1: Paired]
(Data8) TYPE – Bluetooth device type, see below documentation
(Data8) VISIBLE – Visible, [0: Not visible, 1: Visible]

Description

Get search device information. Used to retrieve device information from the search list.

Bluetooth device type list:
0x03 : Bluetooth type Pc
0x04 : Bluetooth type Phone
0x05 : Bluetooth type Brick
0x06 : Bluetooth type Unknown

CMD: FAVOUR__ITEMS = 0x0A

Arguments

(Data8) HARDWARE – See command opCom__Test for details

Return

(Data8) ITEMS – Number of items in favoured list

Description

Get number of devices in favourite. The number of paired devices, not necessary visible or present devices.

CMD: FAVOUR__ITEM = 0x0B

Arguments

(Data8) HARDWARE – See command opCom__Test for details
(Data8) ITEM - Item index within the favour device list
(Data8) LENGTH – Max length of returned string

Return

(Data8) NAME – First character in character string (Zero terminated)
(Data8) PAIRED – Indicate if paired [0: Not paired, 1: Paired]
(Data8) Connected – Indicate if connected [0: Not connected, 1: Paired]
(Data8) TYPE – Bluetooth device type, see SEARCH__ITEM command

Description

Get favour devices information. Used to retrieve device information from the favour list. All devices in the favour list are paired devices.

CMD: GET__ID = 0x0C

Arguments

(Data8) HARDWARE – See command opCom_Test for details
(Data8) LENGTH – Max length of returned string

Return

(Data8) STRING – First character in BT address (Zero terminated)

Description

Get Bluetooth address information.

CMD: GET_BRICKNAME = 0x0D

Arguments

(Data8) LENGTH – Max length of returned string

Return

(Data8) NAME – First character in brick name (Zero terminated)

Description

Get the name of the brick

CMD: GET_NETWORK = 0x0E

Arguments

(Data8) HARDWARE – See command opCom_Test for details
(Data8) LENGTH – Max length of returned string

Return

(Data8) NAME – First character in Access Point name string
(Data8) MAC – First character in MAC address string
(Data8) IP – First character in IP number string

Description

Get network name. Only supported when hardware type is set to WiFi.

CMD: GET_PRESENT = 0x0F

Arguments

(Data8) HARDWARE – See command opCom_Test for details

Return

(Data8) STATUS – Present [0: Not present, 1: Available]

Description

Validate if a valid WiFi dongle are connected to the EV3 P-Brick.

CMD: GET_ENCRYPT = 0x10

Arguments

(Data8) HARDWARE – See command opCom_Test for details
(Data8) ITEM – Index within the favourite list

Return

(Data8) Encryption type, [0: None, 1: WPA2]

Description

Get the used encryption mode for the given WiFi connection.

CMD: GET_INCOMMING = 0x11

Arguments

(Data8) HARDWARE – See command opCom_Test for details
(Data8) LENGTH – Max length of returned string

Return

(Data8) NAME – First character in name

Description

Function enables getting the name of the Bluetooth device which tries to connect.

Instruction	**opCom_Set (CMD, ...)**
Opcode	0xD4
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Can return FAILBREAK
Description	Communication get entry

CMD: SET_ON_OFF = 0x01

Arguments

(Data8) HARDWARE – See command opCom_Test for details
(Data8) ACTIVE – State [0: Disable, 1: Enable]

Description

Enable / disable the given hardware functionality

CMD: SET_VISIBLE = 0x02

Arguments

(Data8) HARDWARE – See command opCom_Test for details
(Data8) VISIBLE – State [0: Invisible, 1: Visible]

Description

Set visibility state. Only valid for Bluetooth hardware.

CMD: SET_SEARCH = 0x03

Arguments

(Data8) HARDWARE – See command opCom_Test for details
(Data8) SEARCH – Search [0: Stop search, 1: Initiate search]

Description

Initiate or stop Bluetooth search scenario.

CMD: SET_PIN = 0x05

Arguments

(Data8) HARDWARE – See command opCom_Test for details
(Data8) NAME – First character in name string
(Data8) PINCODE – First character in pincode string

Description

Set pin code to be used when establishing Bluetooth connection. Used when Bluetooth requests a Pin-code from the user. Not possible to use from running program.

CMD: SET_PASSKEY = 0x06

Arguments

(Data8) HARDWARE – See command opCom_Test for details

(Data8) ACCEPT – Acceptance, [0: Reject, 1: Accept]

Description

Enable responding on a pin-code request used when Bluetooth connection is established using Secure Single Pairing.

CMD: SET_CONNECTION = 0x07

Arguments

(Data8) HARDWARE – See command opCom_Test for details

(Data8) *NAME – First character in device name

(Data8) CONNECTION – Status, [0: Disconnect, 1: Connect]

Description

Control connection status. Initiate or close a Bluetooth connection to a remote Bluetooth device, specified by name.

CMD: SET_BRICKNAME = 0x08

Arguments

(Data8) NAME – First character in brick name

Description

Setting the EV3 brick name. This will be the name shown within the UI of the brick plus the name found when other devices search for this brick using Bluetooth.

CMD: SET_MOVEUP = 0x09

Arguments

(Data8) HARDWARE – See command opCom_Test for details

(Data8) ITEM – Index within table

Description

Enables rearranging the favour list used for WiFi connections. This will move the specified index one step up within the list. Only valid for WiFi favoured list.

CMD: SET_MOVEDOWN = 0x0A

Arguments

(Data8) HARDWARE – See command opCom_Test for details

(Data8) ITEM – Index within table

Description

Enables rearranging the favour list used for WiFi connections. This will move the specified index one step down within the list. Only valid for WiFi favoured list.

CMD: SET_ENCRYPT = 0x0B

Arguments

(Data8) HARDWARE – See command opCom_Test for details

(Data8) ITEM – Index within table

(Data8) ENCRYPT – Specify encryption type [0: None, 1: WPA2]

Description

Enables specifying which encryptions format is requested to be used within the given WiFi connection. Only valid fo WiFi connections.

CMD: SET_SSID = 0x0C

Arguments

(Data8) HARDWARE – See command opCom_Test for details

(Data8) *NAME – Index within table

Description

Enables specifying the SSID name to be used.

4.14 Memory operations

Instruction	**opFile (CMD, …)**
Opcode	0xC0
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	Memory file entry

CMD: OPEN_APPEND = 0x00
Arguments
(Data8) NAME – First character in file name (String, Zero terminated)
Return
(HANDLER) HANDLE – Handle to file
Description
Create file or open for append. Using either '~','.','/' or '.' in front of name indicate that the file is not for the specific folder.

CMD: OPEN_READ = 0x01
Arguments
(Data8) NAME – First character in file name (String, Zero terminated)
Return
(HANDLER) HANDLE – Handle to file
(Data32) SIZE – File size, if file is not found size is set to zero
Description
Open file for read. Using either '~','.','/' or '.' In front of name indicate that the file is not for the specific folder.

CMD: OPEN_WRITE = 0x02
Arguments
(Data8) NAME – First character in file name (String, Zero terminated)
Return
(HANDLER) HANDLE – Handle to file
Description
Create file for write. Using either '~','.','/' or '.' In front of name indicate that the file is not for the specific folder.

CMD: READ_VALUE = 0x03
Arguments
(HANDLER) HANDLE – Handle to file
(Data8) DEL – Delimiter code, see further documentation below
Return
(DataF) VALUE – Value which has been read
Description
Enables reading floating point value from file specified by handle.

Delimiter code supported:
0x00 : No delimiter used
0x01 : Tab used as delimiter
0x02 : Space used as delimiter
0x03 : Return used as delimiter
0x04 : Colon used as delimiter
0x05 : Comma used as delimiter
0x06 : Line feed used as delimiter
0x07 : Return and line feed as delimiter

CMD: WRITE_VALUE = 0x04
Arguments
(HANDLER) HANDLE – Handle to file
(Data8) DEL – Delimiter code, see READ_VALUE command for details
(DataF) VALUE – Value to write
(Data8) FIGURES – Total number of figures inclusive decimal point
(Data8) DECIMALS – Number of decimals
Description
Enables writing floating point value into file specified by handle.

CMD: READ_TEXT = 0x05
Arguments
(HANDLER) HANDLE – Handle to file
(Data8) DEL – Delimiter code, see READ_VALUE command for details
(Data8) LENGTH – Maximal string length to read
Return
(Data8) TEXT – First character in text to read
Description
Enables writing text data into file specified by handle.

CMD: WRITE_TEXT = 0x06
Arguments
(HANDLER) HANDLE – Handle to file
(Data8) DEL – Delimiter code, see READ_VALUE command for details
(Data8) TEXT – First character in text to write (Character string)
Description
Enables writing text data into file specified by handle.

CMD: CLOSE = 0x07
Arguments
(HANDLER) HANDLE – Handle to file
Description
Close file associated with the given handle.

CMD: LOAD_IMAGE = 0x08
Arguments
(Data16) PRGID – Program ID, See below for additional documentation
(Data8) NAME – First character in name (Character string)
Return
(Data32) SIZE– Image size in bytes
(Data32) *IP – Address of image
Description
Enables loading a program from memory

Program ID:
0x00 : Slot reserved for executing user interface program
0x01 : Slot used for executing user projects, apps and tools
0x02 : Slot used for direct command from c_com
0x03 : Slot used for direct command from c_ui
0x04 : Slot used for debug command from ui

CMD: GET_HANDLE = 0x09
Arguments
(Data8) NAME – First character in file name (Character string or handle)
Return
(HANDLER) HANDLE – Handle to file
(Data8) WRITE – Open for write / Append [0: No, 1:Yes]
Description
Enables getting handle from file name. Using either '~','.','/' or '.' In front of name indicate that the file is not for the specific folder.

CMD: MAKE_FOLDER = 0x0A
Arguments
(Data8) NAME – First character in folder name (Character string)
Return
(Data8) SUCCESS – Success, [0: No, 1: Yes]
Description
Enables making a folder if it is not already present.

CMD: GET_POOL = 0x0B

Arguments

(Data32) SIZE – Size of pool

Return

(HANDLER) HANDLE– Handle to pool of memory

(Data32) *IP – Address of image

Description

Enables allocating a pool of memory and get a handle back for accessing this memory pool.

CMD: SET_LOG_SYNC_TIME = 0x0C

Arguments

(Data32) TIME – Sync time to be used during data logging

(Data32) TICK – Sync tick to be used during data logging

Description

Enables setting the sync time and sync tick to be used

CMD: GET_FOLDERS = 0x0D

Arguments

(Data8) NAME – First character in folder name (Ex. "../prjs/")

Return

(Data8) ITEMS – No of sub folders

Description

Enables reading the number of sub folders within a folder

CMD: GET_LOG_SYNC_TIME = 0x0E

Return

(Data32) TIME – Sync time used in data log file

(Data32) TICK – Sync tick used in data log file

Description

Enables reading the sync time and sync tick current used

CMD: GET_SUBFOLDER_NAME = 0x0F

Arguments

(Data8) NAME – First character in folder name (Ex. "../prjs/")

(Data8) ITEM – Sub folder index [1..ITEMS]

(Data8) LENGTH – Maximal string length to read

Return

(Data8) STRING – Fist character of folder name (Character string)

Description

Enables reading sub folder name

CMD: WRITE_LOG = 0x10
Arguments
(HANDLER) HANDLE – Handle to file
(Data32) TIME – Relative time in milliseconds
(Data8) ITEMS – Total number of values in this time slot
(DataF) VALUES – DataF array (handle) containing values
Description
Enables writing time slot samples to file.

CMD: CLOSE_LOG = 0x11
Arguments
(HANDLER) HANDLE – Handle to file
(Data8) NAME – First character in file name (Character string)
Description
Enables closing log file.

CMD: GET_IMAGE = 0x12
Arguments
(Data8) NAME – First character in folder name (Ex. "../prjs/")
(Data16) PRGID – Program ID, see LOAD_IMAGE command
(Data8) ITEM – Sub folder index [1..ITEMS]
Return
(Data32) *IP – Address of image
Description
Enables getting a program image in memory

CMD: GET_ITEM = 0x13
Arguments
(Data8) NAME – First character in folder name (Ex. "../prjs/")
(Data16) STRING – First character in item name string
Return
(Data8) ITEM – Sub folder index [1..ITEMS]
Description
Enables getting index number for a specific item within a folder.

CMD: GET_CACHE_FILES = 0x14
Return
(Data8) ITEM – Number of files in cache
Description
Enables getting the number of files currently in cache.

CMD: PUT_CACHE_FILE = 0x15
Arguments
(Data8) STRING – First character in character string
Description
Enables loading filenames into the "Run recent" memory allocation used for displaying within user interface.

CMD: GET__CACHE__FILE = 0x16
Arguments
(Data8) ITEM – Cache index [1..ITEMS]
(Data8) LENGTH – Maximum string length
Return
(Data8) STRING – First character in character string
Description
Enables getting file name from file in cache and update “Run Recent” within User interface.

CMD: DEL__CACHE__FILE = 0x17
Arguments
(Data8) ITEM – Cache index [1..ITEMS]
(Data8) LENGTH – Maximum string length
Return
(Data8) STRING – First character in character string
Description
Enable deleting file in cache based on file index.

CMD: DEL__SUBFOLDER = 0x18
Arguments
(Data8) NAME – First character in folder name (Ex. “../prjs/”)
(Data8) ITEM – Sub folder index [1..ITEMS]
Description
Enable deleting sub folder

CMD: GET__LOG__NAME = 0x19
Arguments
(Data8) LENGTH – Max string length (Don’t care if NAME is a handle)
(Data8) NAME – First character in file name (Character string or handle)
Description
Enable getting the name on the current open log file.

CMD: OPEN__LOG = 0x1B
Arguments
(Data8) NAME – First character in file name (Character string)
(Data32) SYNCED__TIME –
(Data32) SYNCED__TICK –
(Data32) NOW__TICK –
(Data32) SAMPLE__INTERVAL__IN__MS –
(Data32) DURATION__IN__MS –
(Data8) SDATA – First character in sensor type data (Character string)
Return
(HANDLER) HANDLE – Handle to file
Description
Enable creating a file for data logging. Using either ‘~’,’,’,’/’ or ‘.’ In front of name indicate that the file is not for the specific folder.

CMD: READ_BYTES = 0x1C

Arguments

(HANDLER) HANDLE – Handle to file

(Data16) BYTES – Number of bytes to read

Return

(Data8) DESTINATION – First byte in byte stream to be read

Description

Enable reading a number of bytes from file specified by handle.

CMD: WRITE_BYTES = 0x1D

Arguments

(HANDLER) HANDLE – Handle to file

(Data16) BYTES – Number of bytes to write

(Data8) SOURCE – First byte in byte stream to write

Description

Enable writing a number of bytes to file specified by handle.

CMD: REMOVE = 0x1E

Arguments

(HANDLER) HANDLE – Handle to file

Description

Delete file associated with the given handle. Using either '~','.','/' or '.' in front of name indicate that the file is not for the specific folder.

CMD: MOVE = 0x1F

Arguments

(Data8) SOURCE – First character in source file name

(Data8) DESTINATION – First character in destination file name

Description

Enable moving a file from source to destination. Using either '~','.','/' or '.' in front of name indicate that the file is not for the specific folder. Both source and destination name is expected as character strings.

Instruction	**opArray (CMD, ...)**
Opcode	0xC1
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Can change to BUSYBREAK or FAILBREAK
Description	Array entry

CMD: DELETE = 0x00
Arguments
(HANDLER) HANDLE – Array handle
Description
Enable deleting array based on array handle.

CMD: CREATE8 = 0x01
Arguments
(Data32) ELEMENTS – Number of elements
Return
(HANDLER) HANDLE – Array handle
Description
Enable creating an array with data type 8-bit integer.

CMD: CREATE16 = 0x02
Arguments
(Data32) ELEMENTS – Number of elements
Return
(HANDLER) HANDLE – Array handle
Description
Enable creating an array with data type 16-bit integer.

CMD: CREATE32 = 0x03
Arguments
(Data32) ELEMENTS – Number of elements
Return
(HANDLER) HANDLE – Array handle
Description
Enable creating an array with data type 32-bit integer.

CMD: CREATEF = 0x04
Arguments
(Data32) ELEMENTS – Number of elements
Return
(HANDLER) HANDLE – Array handle
Description
Enable creating an array with data type floating point.

CMD: RESIZE = 0x5

Arguments

(HANDLER) HANDLE – Array handle

(Data32) ELEMENTS – Total number of elements (In new array)

Description

Enable resizing an existing array. Number of elements equals the total number of element required in the new array.

CMD: FILL = 0x06

Arguments

(HANDLER) HANDLE – Array handle

(type) Value – Value to write (Type depends on type of array)

Description

Enable filling an array with predefined values.

CMD: COPY = 0x07

Arguments

(HANDLER) H__SOURCE – Source array handle

(HANDLER) H__DESITANTION– Destination array handle

Description

Enable copying one array into another array.

CMD: INIT8 = 0x08

Arguments

(HANDLER) HANDLE – Array handle

(Data32) INDEX – Index to element to write

(Data32) ELEMENTS – Number of elements to write

(Data8) VALUE – First value to write

Description

Enable writing either one or multiple (Allocated continuously in memory) values into array.

CMD: INIT16= 0x09

Arguments

(HANDLER) HANDLE – Array handle

(Data32) INDEX – Index to element to write

(Data32) ELEMENTS – Number of elements to write

(Data16) VALUE – First value to write

Description

Enable writing either one or multiple (Allocated continuously in memory) values into array.

CMD: INIT32 = 0x0A

Arguments

(HANDLER) HANDLE – Array handle
(Data32) INDEX – Index to element to write
(Data32) ELEMENTS – Number of elements to write
(Data32) VALUE – First value to write

Description

Enable writing either one or multiple (Allocated continuously in memory) values into array.

CMD: INITF = 0x0B

Arguments

(HANDLER) HANDLE – Array handle
(Data32) INDEX – Index to element to write
(Data32) ELEMENTS – Number of elements to write
(DataF) VALUE – First value to write

Description

Enable writing either one or multiple (Allocated continuously in memory) values into array.

CMD: SIZE = 0x0C

Arguments

(HANDLER) HANDLE – Array handle

Return

(Data32) ELEMENTS – Number of bytes in array

Description

Enable reading the total number of elements within an array.

CMD: READ_CONTENT = 0x0D

Arguments

(Data16) PRGID – Program slot (Must be running)
(HANDLER) HANDLE – Array handle
(Data32) INDEX – Index to first element to read
(Data32) BYTES – Number of elements to read

Return

(Data8) ARRAY – First byte of array to receive data

Description

Enable reading data from array currently active within running program.

CMD: WRITE__CONTENT = 0x0E
Arguments
(Data16) PRGID – Program slot (Must be running)
(HANDLER) HANDLE – Array handle
(Data32) INDEX – Index to first element to write
(Data32) BYTES – Number of elements to write
(Data8) ARRAY – First byte of array to deliver data
Description
Enable writing data into array current used by the running program.

CMD: READ__SIZE = 0x0F
Arguments
(Data16) PRGID – Program slot (Must be running)
(HANDLER) HANDLE – Array handle
Return
(Data32) BYTES – Number of bytes in array
Description
Enable reading the size of array current used within running program.

Instruction	**opArray__Write (HANDLE, INDEX, VALUE)**
Opcode	0xC2
Arguments	(HANDLER) HANDLE – Array handle (Data32) INDEX – Index to element to write (type) VALUE - Value to write. Type depends on array type.
Dispatch status	Can change to FAILBREAK
Description	This function enables writting data into array from within a running program.

Instruction	**opArray__Read (HANDLE, INDEX, VALUE)**
Opcode	0xC3
Arguments	(HANDLER) HANDLE – Array handle (Data32) INDEX – Index to element to read
Return	(type) VALUE - Value to read. Type depends on array type.
Dispatch status	Can change to FAILBREAK
Description	This function enables reading data from an array within a running program.

Instruction	**opArray__Append (TOTAL, FREE)**
Opcode	0xC4
Arguments	(HANDLER) HANDLE – Array handle (type) VALUE – New element value to append. Type depends on array type.
Dispatch status	Can change to FAILBREAK
Description	This function enables appending one value to existing array.

Instruction	**opMemory_Usage (TOTAL, FREE)**
Opcode	0xC5
Arguments	(Data32) TOTAL – Total amount of internal memory [KB] (Data32) FREE – Free memory [KB]
Dispatch status	Unchanged
Description	This function calculated the amount of used and free user memory.

Instruction	**opFilename (CMD, ...)**
Opcode	0xC6
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	Memory filename entry

CMD: EXIST = 0x10

Arguments

(DATA8) NAME – First character in file name (Character string)

Return

(Data8) FLAG – Exist, (0: No, 1: Yes)

Description

Enable testing if file exists. Using either '~','.','/' or '.' in front of name indicate that the file is not for the specific folder.

CMD: TOTALSIZE = 0x11

Arguments

(DATA8) NAME – First character in file name (Character string)

Return

(Data32) FILES – Total number of files

(Data32) SIZE – Total folder size, [KB]

Description

Enable calculating folder/file size. Using either '~','.','/' or '.' in front of name indicate that the file is not for the specific folder.

CMD: SPLIT = 0x12

Arguments

(DATA8) FILENAME – First character in file name (Character string)

"../folder/subfolder/name.txt"

(Data8) LENGTH – Maximum length for each of below parameters

(Data8) FOLDER – First character in folder name (Character string)

"../folder/subfolder"

Return

(Data8) NAME - First character in name (Character string) "name"

(Data8) EXT - First character in extension (Character string) ".txt"

Description

Enable splitting filename into folder, name and extension

CMD: MERGE = 0x13
Arguments
(Data8) FOLDER – First character in folder name (Character string)
"../folder/subfolder"
(Data8) NAME - First character in name (Character string) "name"
(Data8) EXT - First character in extension (Character string) ".txt"
(Data8) LENGTH – Maximum length for each of above parameters
Return
(DATA8) FILENAME – First character in file name (Character string)
"../folder/subfolder/name.txt"
Description
Enable merging folder, name and extension into filename

CMD: CHECK = 0x14
Arguments
(DATA8) FILENAME – First character in file name (Character string)
"../folder/subfolder/name.txt"
Return
(Data8) STATUS – Filename ok, [0: Fail, 1: Ok]
Description
Enable check a filename including folder structure

CMD: PACK = 0x15
Arguments
(DATA8) FILENAME – First character in file name (Character string)
"../folder/subfolder/name.txt"
Description
Enable packing file or folder into "raf" container

CMD: UNPACK = 0x16
Arguments
(DATA8) FILENAME – First character in file name (Character string)
"../folder/subfolder/name"
Description
Enable unpacking "raf" container

CMD: GET__FOLDERNAME = 0x17
Arguments
(DATA8) LENGTH – Maximum length for returned folder name
Return
(Data8) FOLDERNAME First character in folder name (Character string)
"../folder/subfolder"
Description
Enable getting current folder name

4.15 User interface operations

Instruction	**opUI_DRAW (CMD, …)**
Opcode	0x84
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Can change to BUSYBREAK
Description	User interface draw entry

CMD: UPDATE = 0x00
Description
Automatically triggers a refreshes of the display.

CMD: PIXEL = 0x02
Arguments
(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X point, [0 - 177]
(Data16) Y0 – Specify Y point, [0 - 127]
Description
Enable drawing a single pixel.

CMD: LINE = 0x03
Arguments
(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) X1 – Specify X end point
(Data16) Y1 – Specify Y end point
Description
Enable drawing a line between above coordinates

CMD: CIRCLE = 0x04
Arguments
(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) R – Radius for circle
Description
Enable drawing a circle

CMD: TEXT = 0x05
Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data8) STRING – First character in string to draw (Zero terminated)

Description

Enable displaying text

CMD: ICON = 0x06

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data8) TYPE – Icon type (Selection of internal available icons)
(Data) NO – Icon number

Description

Enable displaying predefined icons

CMD: PICTURE = 0x07

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data32) *IP – Address on picture

Description

Enable displaying picture from internal memory

CMD: VALUE = 0x08

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(DataF) VALUE – Floating point value to display
(Data8) FIGURES – Total number of figures inclusive decimal point
(Data8) DECIMALS – Number of decimals

Description

Enable displaying floating point value with specified resolution

CMD: FILLRECT = 0x09

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) X1 – Specify X size
(Data16) Y1 – Specify Y size

Description

Enable drawing a filled rectangle

CMD: RECT = 0x0A

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) X1 – Specify X size
(Data16) Y1 – Specify Y size

Description

Enable drawing a rectangle

CMD: NOTIFICATION = 0x0B

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data8) ICON1 – First icon
(Data8) ICON2 – Second icon
(Data8) ICON3 – Third icon
(Data8) STRING – First character in notification string
(Data8) *STATE – State, 0 = INIT

Description

Enable displaying multiple notification options to the user

CMD: QUESTION = 0x0C

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data8) ICON1 – First icon
(Data8) ICON2 – Second icon
(Data8) STRING – First character in question string
(Data8) *STATE – State, 0 = No, 1 = OK

Return

(Data8) OK – Answer, 0 = No, 1 = OK, -1 = SKIP

Description

Enable displaying question to the user and returning the users selection.

CMD: KEYBOARD = 0x0D

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data8) LENGTH – Maximum string length
(Data8) DEFAULT – Default string, (0 = None)
(Data8) *CHARSET – Internal use
(Must be a variable initialised by a “valid character set”)

Return

(Data8) STRING – First character in string received from keyboard input

Description

Enable displaying a keyboard to the user, either with or without a pre-defined text. The function will return the input from keyboard.

CMD: BROWSE = 0x0E

Arguments

(DATA8) TYPE – See further specification below in type
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) X1– X size
(Data16) Y1 – Y size
(Data8) LENGTH – Maximum string length

Return

(Data8) TYPE – Item type (Folder, byte code file, sound file,…)
(Must be a zero initialized variable)
(Data8) STRING – First character in string receiving selected item name

Description

Enable displaying different browsers and content.

TYPE available:
0x00 : Browser for folders
0x01 : Browser for folders and files
0x02 : Browser for cached / recent files
0x03 : Browser for files

CMD: VERTBAR = 0x0F

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) X1 – Specify X end point
(Data16) Y1 – Specify Y end point
(Data16) MIN – Minimum value
(Data16) MAX – Maximum value
(Datat16) ACT – Actual value

Description

Enable display a vertical bar including a fill portion depending on the actual value in relation to minimum and maximum limits specified.

CMD: INVERSERECT = 0x10

Arguments

(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) X1 – Specify X size
(Data16) Y1 – Specify Y size

Description

Enable inverting a already drawn filled rectangle.

CMD: SELECT_FRONT = 0x11

Arguments

(DATA8) TYPE – Specify font type, [0 - 2]

Description

Enable specifying the font to be used. Font will change to 0 when UPDATE is called.

CMD: TOPLINE = 0x12

Arguments

(DATA8) ENABLE – Enable or disable top status line, [0: Disable, 1: Enable]

Description

Enable specifying the font to be used. Font will change to 0 when UPDATE is called.

CMD: FILLWINDOW = 0x13

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) Y1 – Specify Y size

Description

Enable filling part of the display within the horizontal plane.

CMD: DOTLINE = 0x15

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) X1 – Specify X end point
(Data16) Y1 – Specify Y end point
(Data16) ON – On pixels
(Data16) OFF – Off pixels

Description

Enable drawing a dotted line between two coordinates

CMD: VIEW__VALUE = 0x16

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(DataF) VALUE – Floating point value to display
(Data8) FIGURES – Total number of figures inclusive decimal point
(Data8) DECIMALS – Number of decimals

Description

Enable displaying floating point value with specified resolution

CMD: VIEW__UNIT = 0x17

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(DataF) VALUE – Floating point value to display
(Data8) FIGURES – Total number of figures inclusive decimal point
(Data8) DECIMALS – Number of decimals
(Data8) LENGTH – Maximum string length
(Data8) STRING – First character in string to draw

Description

Enable displaying floating point value with specified resolution plus with specified units attached.

CMD: FILLCIRCLE = 0x18

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) R – Radius for circle

Description

Enable drawing a circle

CMD: STORE = 0x19

Arguments

(DATA8) NO – Level number

Description

Enable storing current display image in memory.

CMD: RESTORE = 0x1A

Arguments

(DATA8) NO – Level number (N=0 => Saved screen just before run)

Description

Enable restoring previous display window.

CMD: ICON_QUESTION = 0x1B

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data8) *STATE – State, 0 = INIT
(Data32) ICONS – Bitfield with icons

Description

Enable displaying a question window using different icon

CMD: BMPFILE = 0x1C

Arguments

(DATA8) Color – Specify either black or white, [0: White, 1: Black]
(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data8) NAME – First character in filename (Character string)

Description

Enable displaying BMP file from icon file within running project.

CMD: GRAPH_SETUP = 0x1E

Arguments

(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data8) ITEMS – Number of datasets in arrayes
(Data16) OFFSET – Data16 array (handle) containing Y point
(Data16) SPAN – Data16 array (handle) containing Y size
(DataF) MIN – DataF array (handle) containing min values
(DataF) MAX – DataF array (handle) containing max values
(DataF) SAMPLE – DataF array (handle) containing sample values

Description

Enable continuously drawing a graph including updating the graph according to new data value.

CMD: GRAPH_DRAW = 0x1F

Arguments

(Data8) VIEW – Dataset number to view (0 = All)

Description

Enable live updating graph window including scroll.

CMD: TEXTBOX = 0x20

Arguments

(Data16) X0 – Specify X start point, [0 - 177]
(Data16) Y0 – Specify Y start point, [0 - 127]
(Data16) X1 – Specify X end point
(Data16) Y1 – Specify Y end point
(Data8) TEXT – First character in text box (Must be zero terminated)
(Data32) SIZE – Maximal text size (Including zero termination)
(Data8) DEL – Delimiter code, see below documentation

Return

(Data16) LINE – Selected line number

Description

Draws and controls a text box (one long string containing characters and line delimiters) on the screen.

Delimiter code supported:
0x00 : No delimiter used
0x01 : Tab used as delimiter
0x02 : Space used as delimiter
0x03 : Return used as delimiter
0x04 : Colon used as delimiter
0x05 : Comma used as delimiter
0x06 : Line feed used as delimiter
0x07 : Return and line feed as delimiter

Instruction	**opUI_Flush ()**
Opcode	0x80
Arguments	
Dispatch status	Unchanged
Description	This function flushes user interface buffers.

Instruction	**opUI_Read (CMD, ...)**
Opcode	0x81
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Can change to BUSYBREAK or FAILBREAK
Description	User interface read entry

CMD: GET_VBATT = 0x01

Return

(DataF) Value – Battery voltage [V]

CMD: GET_IBATT = 0x02

Return

(DataF) Value – Battery current [A]

CMD: GET__OS__VERS = 0x03
Arguments
(Data8) LENGTH – Maximal length of string to be returned
Return
(Data) DESTINATION – String variable or handle to string
Description
Enable getting OS version string

CMD: GET__EVENT = 0x04
Return
(Data8) EVENT– Event, [1,2 = Bluetooth event], Internal use

CMD: GET__TBATT = 0x05
Return
(DataF) Value – Battery temperature rise [C]

CMD: GET__IMOTOR = 0x07
Return
(DataF) Value – Motor current [A]

CMD: GET__STRING = 0x08
Arguments
(Data8) LENGTH – Maximal length of string to be returned
Return
(Data) DESTINATION – String variable or handle to string
Description
Enable getting a string from the terminal

CMD: GET__HW__VERS = 0x09
Arguments
(Data32) LENGTH – Maximal length of string returned (-1 = No check)
Return
(Data8) DESTINATION – String variable or handle to string
Description
Enable reading the hardware version on the given hardware

CMD: GET__FW__VERS = 0x0A
Arguments
(Data32) LENGTH – Maximal length of string returned (-1 = No check)
Return
(Data8) DESTINATION – String variable or handle to string
Description
Enable reading the firmware version currently on the EV3 brick

CMD: GET__FW__BUILD = 0x0B
Arguments

(Data32) LENGTH – Maximal length of string returned (-1 = No check)

Return

(Data8) DESTINATION – String variable or handle to string

Description

Enable reading the OS version currently on the EV3 brick

CMD: GET_OS_BUILD = 0x0C

Arguments

(Data32) LENGTH – Maximal length of string returned (-1 = No check)

Return

(Data8) DESTINATION – String variable or handle to string

Description

Enable reading the OS build info currently on the EV3 brick

CMD: GET_ADDRESS = 0x0D

Arguments

(Data32) VALUE – Address from lms_cmdin

Description

Get address from terminal, Command used for advanced debugging.

CMD: GET_CODE = 0x0E

Arguments

(Data32) LENGTH – Maximal code stream length

Return

(Data32) *IMAGE – Address of image

(Data32) *GLOBAL – Address of global variables

(Data8) FLAG – Flag tells if image is ready to execute, [1: Ready]

Description

Get code from terminal, Command used for advanced debugging.

CMD: KEY = 0x0F

Arguments

(Data8) VALUE – Key from lms_cmdin (0 = no key)

Description

Get key from terminal. Command used for advanced debugging.

CMD: GET_SHUTDOWN= 0x10

Return

(Data8) FLAG – Flag [Want to shutdown]

Description

Read warning bit Get and clear shutdown flag (Internal use).

CMD: GET_WARNING = 0x11

Return

(Data8) WARNING – Bit field containing various warnings

Description

Read warning bit field.

Warnings:
0x01 : Warning temperature
0x02 : Warning current
0x04 : Warning voltage
0x08 : Warning memory
0x10 : Warning DSPSTAT
0x20 : Warning RAM
0x40 : Warning battery low
0x80 : Warning busy
0x3F : Warnings

CMD: GET_LBATT = 0x12

Return

(Data8) PCT – Battery level in percentage [0 - 100]

CMD: TEXTBOX_READ = 0x15

Arguments

(Data8) TEXT – First character in text box text (Must be zero terminated)
(Data32) SIZE – Maximal text size
(Data8) DEL – Delimiter code
(Data8) LENGTH- Maximal length of string returned (-1 = No Check)
(Data16) LINE – Selected line number

Return

(Data8) DESTINATION – String variable or handle to string

Description

Enable reading line from text box

CMD: GET_VERSION = 0x1A

Arguments

(Data32) LENGTH – Maximal length of string returned (-1 = No check)

Return

(Data8) DESTINATION – String variable or handle to string

Description

Enable reading the hardware version on the given hardware

CMD: GET_IP = 0x1B

Arguments

(Data32) LENGTH – Maximal length of string returned (-1 = No check)

Return

(Data8) DESTINATION – String variable or handle to string

Description

Enable reading the IP address when available

CMD: GET_SDCARD= 0x1D

Return
(Data8) STATE – SD Card present, [0: No, 1: Present]
(Data32) TOTAL – SD Card memory size [KB]
(Data32) FREE – Amount of free memory [KB]

CMD: GET_USBSTICK = 0x1E
Return
(Data8) STATE – USB memory stick present, [0: No, 1: Present]
(Data32) TOTAL – USB memory size [KB]
(Data32) FREE – Amount of free memory [KB]

CMD: GET_FW_BUILD = 0x1F
Arguments
(Data32) LENGTH – Maximal length of string returned (-1 = No check)
Return
(Data8) DESTINATION – String variable or handle to string

Instruction	**opUI_WRITE (CMD, ...)**
Opcode	0x82
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Can change to BUSYBREAK or FAILBREAK
Description	User interface write entry

CMD: WRITE_FLUSH = 0x01
Description
Enable updating the terminal with all latest data.

CMD: FLOATVALUE = 0x02
Arguments
(DataF) VALUE – Value to write
(Data8) FIGURES – Total number figures inclusive decimal point
(Data8) DECIMALS – Number of decimals
Description
Enable writing a floating point value

CMD: PUT_STRING = 0x08
Arguments
(Data8) STRING – First character in string to write
Description
Enable writing a string

CMD: VALUE8 = 0x09
Arguments
(Data8) VALUE – Value to write
Description
Enable writing a 8-bit integer

CMD: VALUE16 = 0x0A

Arguments

(Data8) VALUE – Value to write

Description

Enable writing a 16-bit integer

CMD: VALUE32 = 0x0B

Arguments

(Data8) VALUE – Value to write

Description

Enable writing a 32-bit integer

CMD: VALUEF= 0x0C

Arguments

(Data8) VALUE – Value to write

Description

Enable writing a floating point

CMD: DOWNLOAD_END = 0x0F

Description

Enables updating the user interface browser when a file download is done. A small sound will also be played to indicate an update has happened.

CMD: SCREEN_BLOCK = 0x10

Arguments

(Data8) STATUS – Value [o: Normal, 1: Blocked]

Description

Enable setting or clearing screen block status (With screen blocked – all graphics action are disabled.

CMD: TEXTBOX_APPEND = 0x15

Arguments

(Data8) TEXT – First character in text box text (Must be zero terminated)
(Data32) SIZE – Maximal text size (Including zero termination)
(Data8) DEL – Delimiter code
(Data8) SOURCE – String variable or handle to string append

Description

Enable appending line of text at the bottom of a text box

CMD: SET_BUSY = 0x16

Arguments

(Data8) VALUE – Set busy value, [0, 1]

Description

Enable setting the busy flag.

CMD: SET_TESTPIN = 0x18

Arguments

(Data8) STATE – Set test-pin, [0, 1]

Description

Enable controlling the test-pin. Is only possible within test mode.

CMD: INIT__RUN= 0x19

Description

Start the “MINDSTORMS” “Run” screen

CMD: LED = 0x1B

Arguments

(Data8) PATTERN – LED pattern, see below for more documentation

Description

Enable controlling the LED light around the button on EV3

LED Patterns:

0x00 : Led off
0x01 : Led green
0x02 : Led red
0x03 : Led orange
0x04 : Led green flashing
0x05 : Led red flashing
0x06 : Led orange flashing
0x07 : Led green pulse
0x08 : Led red pulse
0x09 : Led orange pulse

CMD: POWER = 0x1D

Arguments

(Data8) VALUE – Set power value, [0, 1]

Description

Enable setting the power flag.

CMD: GRAPH__SAMPLE = 0x1E

Description

Update tick to scroll graph horizontal in memory when drawing graph in “Scope” mode

CMD: TERMINAL = 0x1F

Arguments

(Data8) STATE – Enable / disable terminal, [0, 1]

Description

Enable or disable terminal port.

Opcode	0x83
Arguments	(Data8) CMD => Specific command parameter documented below
Dispatch status	Unchanged
Description	User interface button entry

CMD: SHORTPRESS = 0x01
Arguments
(Data8) BUTTON – Button to evaluate, see below documentation
Return
(Data8) STATE – Button status, [0: No, 1: Yes]
Description
Enable verifying if a short button press has happen.

LED Patterns:
0x00 : No button
0x01 : Up button
0x02 : Enter button
0x03 : Down button
0x04 : Right button
0x05 : Left button
0x06 : Back button
0x07 : Any button

CMD: LONGPRESS = 0x02
Arguments
(Data8) BUTTON – Button to evaluate, reference SHORTPRESS command
Return
(Data8) STATE – Button status, [0: No, 1: Yes]
Description
Enable verifying if a button has been pressed for long period.

CMD: WAIT_FOR_PRESS = 0x03
Description
Enable waiting for any button press.

CMD: FLUSH = 0x04
Description
Enable removing all previous button status..

CMD: PRESS = 0x05
Arguments
(Data8) BUTTON – Button to evaluate, reference SHORTPRESS command
Description
Enable activating a button press from software.

CMD: RELEASE = 0x06
Arguments
(Data8) BUTTON – Button to evaluate, reference SHORTPRESS command
Description
Enable activating a button release from software.

CMD: GET__HORZ = 0x07
Return
(Data16) VALUE – Horizontal arrows data (-1: Left, +1: Right, 0: Not pressed)
Description
Enable reading current arrow position within the horizontal plane.

CMD: GET__VERT = 0x08
Return
(Data16) VALUE – Vertical arrows data (-1: Left, +1: Right, 0: Not pressed)
Description
Enable reading current arrow position within the vertical plane.

CMD: PRESSED = 0x09
Arguments
(Data8) BUTTON – Button to evaluate, reference SHORTPRESS command
Return
(Data8) STATE – Button status, [0: No, 1: Yes]
Description
Enable verifying if a button is being pressed.

CMD: SET__BACK__BLOCK = 0x0A
Argument
(Data8) BLOCKED – Set UI back button blocked flag
(0: Not blocked, 1: Blocked)
Description
Enable blocking the back button functionality.

CMD: GET__BACK__BLOCK = 0x0B
Return
(Data8) BLOCKED – Get UI back button blocked flag
(0: Not blocked, 1: Blocked)
Description
Enable reading the back button block flag.

CMD: TESTSHORTPRESS = 0x0C
Arguments
(Data8) BUTTON – Button to evaluate, reference SHORTPRESS command
Return
(Data8) STATE – Button status, [0: No, 1: Yes]
Description
Enable verifying if a button has been/are being pressed for short duration.

CMD: TESTLONGPRESS = 0x0D
Arguments
(Data8) BUTTON – Button to evaluate, reference SHORTPRESS command
Return
(Data8) STATE – Button status, [0: No, 1: Yes]
Description
Enable verifying if a button has been/are being pressed for long duration.

CMD: GET_BUMBED = 0x0E
Arguments
(Data8) BUTTON – Button to evaluate, reference SHORTPRESS command
Return
(Data8) STATE – Button status, [0: No, 1: Yes]
Description
Enable verifying if a button has been bumped = Pressed and released.

CMD: GET_CLICK = 0x0F
Return
(Data8) CLICK – Click sound requested, [0: No, 1: Yes]
Description
Get and clear click sound request.

Instruction	**opKeep_Alive (MINUTES)**
Opcode	0x90
Arguments	(Data8) MINUTES – Number of minutes before entering sleep mode
Dispatch status	Unchanged
Description	This function enables controlling the power down time.

5 Device type list

The EV3 firmware uses a build-in device type list in order to know how individual data from the various external devices are to be read and understood.

Below list includes part of information within the device type list. More details can be found within the actual firmware source which is release under the GNU GPL V2 license.

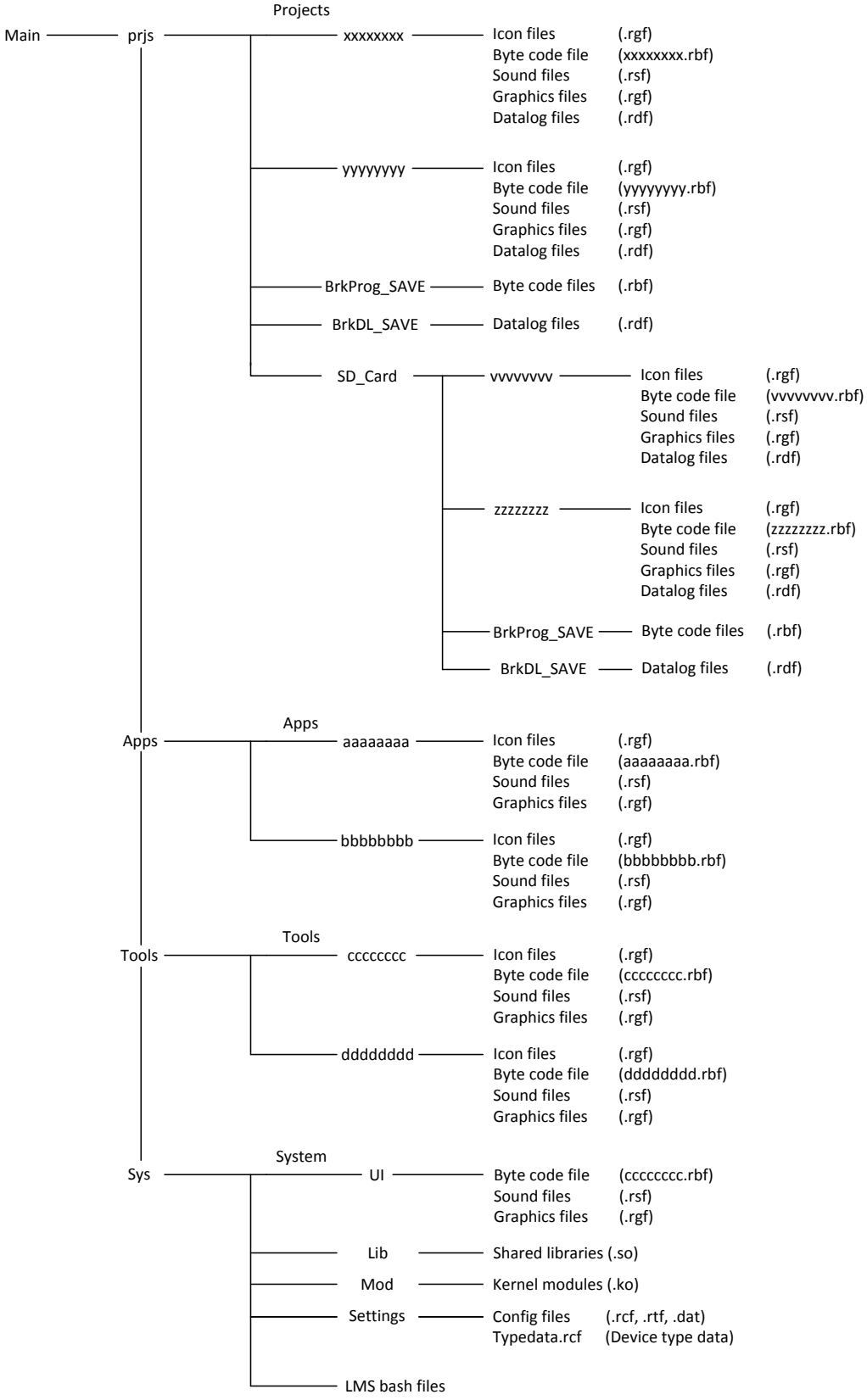
TYPE	MODE	NAME
1	0	NXT-Touch
	1	NXT-Bump
2	0	NXT-Light-Reflected
	1	NXT-light-Ambient
3	0	NXT-Sound-DB
	1	NXT-Sound-DBA
4	0	NXT-Color-Reflected
	1	NXT-Color-Ambient
	2	NXT-Color-Color
	3	NXT-Color-Green
	4	NXT-Color-Blue
	5	NXT-Color-Raw
5	0	NXT-Ultrasonic-Cm
	1	NXT-Ultrasonic-Inch
6	0	NXT-Temperature-C
	1	NXT-Temperature-F
7	0	EV3-Large-Motor-Degree
	1	EV3-Large-Motor-Rotation
	2	EV3-Large-Motor-Power
8	0	EV3-Medium-Motor-Degree
	1	EV3-Medium-Motor-Rotation
	2	EV3-Medium-Motor-Power
9		Free
...	...	...
13		Free
14	0	Output for 3th party devices
15		Free
16	0	EV3-Touch
	1	EV3-Bump
17		Free
...	...	...
20		Free
21	0	Test purpose
22		Free
...	...	...
27		Free
28	0	3th party input , 1 mode, Scale 0 - 4095
	1	3th party input , 2 mode, Scale 0 - 5000
	2	3th party input , 3 mode, Scale 0 - 10000
	3	3th party input , 4 mode, Scale 0 - 20000

TYPE	MODE	NAME

29	0	EV3-Color-Reflected
	1	EV3-Color-Ambient
	2	EV3-Color-Color
	3	EV3-Color-Reflected-Raw
	4	EV3-Color-RGB-Raw
	5	EV3-Color-Calibration
30	0	EV3-Ultrasonic-Cm
	1	EV3-Ultrasonic-Inch
	2	EV3-Ultrasonic-Listen
	3	EV3-Ultrasonic-SI-Cm
	4	EV3-Ultrasonic-SI-Inch
	5	EV3-Ultrasonic-DC-Cm
	6	EV3-Ultrasonic-DC-Inch
31		Free
32	0	EV3-Gyro-Angle
	1	EV3-Gyro-Rate
	2	EV3-Gyro-Fast
	3	EV3-Gyro-Rate & Angle
	4	EV3-Gyro-Calibration
33	0	EV3-IR-Proximity
	1	EV3-IR-Seeker
	2	EV3-IR-Remote
	3	EV3-IR-Remote-Advanced
	4	Not utilized
	5	EV3-IR-Calibration
34		Free
...	...	...
98		Free
99	0	Energy-Meter-Voltage-In
	1	Energy-Meter-Amps-In
	2	Energy-Meter-Voltage-Out
	3	Energy-Meter-Amps-Out
	4	Energy-Meter-Joule
	5	Energy-Meter-Watts-In
	6	Energy-Meter-Watts-Out
	7	Energy-Meter-All
100	0	IIC-Byte
	1	IIC-WORD
101	0	NXT-Test
102		Free
...	...	...
120		Free

6 Folder structure within firmware

Below figure illustrates the folder structure used within firmware. Below structure needs to be followed to allow the build functionality to show and use downloaded files.



7 Program examples

Below are a few low-level program examples using the program structure documented in the earlier sections. These program example are not too been evaluated as final programs but more as examples demonstrating how a program could be build / structured.

7.1 Line follower

Simple exmple that shows how to build a line following program.

```
UBYTE prg[] =                                   //  p5.c
{                                               //
    PROGRAMHeader(0,1,0),                       // 0    VersionInfo,NumberOfObjects,GlobalBytes
    VMTHREADHeader(0,3),                        // 12   OffsetToInstructions,LocalBytes
                                                //
                                                // 24  VMTHREAD1
                                                //      {
    opUI_WRITE,LC0(PUT_STRING),LCS,             //        UI_WRITE(STRING,"    LineFollower\r\n")
      ' ',' ',' ',' ',' ',                      //
      'L','i','n','e','F','o','l',             //
      'l','o','w','e','r',                      //
       '\r','\n',0,                             //
    opUI_FLUSH,                                 //        UI_FLUSH
                                                //
    opINPUT_READ,LC0(0),LC0(1),                 //        INPUT_READ(0,0,2,LV0)       // Read light sensor
    LC1(2),LC0(1),LV0(0),                       //
    opDIV8,LV0(0),LC0(2),LV0(0),                //        LV0 /=  2
    opADD8,LV0(0),LC0(15),LV0(0),               //        LV0 +=  15
                                                //        do
                                                //        {
    opINPUT_READ,LC0(0),LC0(1),                 //          INPUT_READ(0,0,2)         // Read light sensor
    LC1(2),LC0(1),LV0(1),                       //
    opSUB8,LV0(1),LV0(0),LV0(1),                //          LV1    -=  LV0
    opDIV8,LV0(1),LC0(2),LV0(1),                //          LV1    /=  2
    opADD8,LC1(40),LV0(1),LV0(2),               //          LV2     =  40 + LV1
    opSUB8,LC1(40),LV0(1),LV0(3),               //          LV3     =  40 - LV1
                                                //
    opOUTPUT_POWER,LC0(0),LC0(1),LV0(3),        //          OUTPUT_POWER(0,0x1,LV3)   // Motor 1 = LV3
    opOUTPUT_POWER,LC0(0),LC0(2),LV0(2),        //          OUTPUT_POWER(0,0x2,LV2)   // Motor 2 = LV2
    opOUTPUT_START,LC0(0),LC0(3),               //          OUTPUT_START(0,0x3)
                                                //        }
    opJR,LC1(-39),                              //        while (TRUE)
    opOBJECT_END                                //      }
};
```

7.2 Scheduling blocks

Program that shows how individual program blocks can trigger other program block through a program execution process.

```
//  (VMTHREAD1--DELAY(2)--TRIG(2)---------------------TRIG(3)----------------WAIT(3)--WAIT(2)------)
//                          \                          \                          /        /
//                           \                          (BLOCK3----DELAY(2)--)        /
//                            \                        /                          /
//                             (BLOCK2--DELAY(2)--TRIG(3)---------------DELAY(6)----------)
```

```
UBYTE     prg[] =                                                    // p6.c
{                                                                    //
    PROGRAMHeader(0,3,0),                                            // VersionInfo, NumberOfGlobal
                                                                     // GlobalBytes
    VMTHREADHeader(52,12),                                           // OffsetToInstructions,LocalBytes
    BLOCKHeader(192,1,1),                                            // OffsetToInstructions,
                                                                     // OwnerObjectId, TriggerCount
    BLOCKHeader(266,1,2),                                            // OffsetToInstructions
                                                                     // OwnerObjectId,TriggerCount
                                                                     //
                                                                     //      VMTHREAD1
                                                                     //      {
    opUI_WRITE,LC0(PUT_STRING),LCS,'B','L','O','C','K','1','\r','\n',0,  //      UI_WRITE(STRING,"BLOCK1\r\n")
    opUI_FLUSH,                                                      //      UI_FLUSH
                                                                     //
    opUI_WRITE,LC0(PUT_STRING),LCS,'D','E','L','A','Y','(','2',')','\r','\n',0,//      UI_WRITE(STRING,"DELAY(2)\r\n")
    opUI_FLUSH,                                                      //      UI_FLUSH
    opTIMER_WAIT,LC2(2000),LV0(0),                                   //      DELAY(2)
    opTIMER_READY,LV0(0),                                            //
                                                                     //
    opUI_WRITE,LC0(PUT_STRING),LCS,'T','R','I','G','(','2',')','\r','\n',0,  //      UI_WRITE(STRING,"TRIG(2)\r\n")
    opUI_FLUSH,                                                      //      UI_FLUSH
    opOBJECT_TRIG,LC0(2),                                            //      TRIG(2)
                                                                     //
    opUI_WRITE,LC0(PUT_STRING),LCS,'T','R','I','G','(','3',')','\r','\n',0,  //      UI_WRITE(STRING,"TRIG(3)\r\n")
    opUI_FLUSH,                                                      //      UI_FLUSH
    opOBJECT_TRIG,LC0(3),                                            //      TRIG(3)
                                                                     //
    opUI_WRITE,LC0(PUT_STRING),LCS,'W','A','I','T','(','3',')','\r','\n',0,  //      UI_WRITE(STRING,"WAIT(3)\r\n")
    opUI_FLUSH,                                                      //      UI_FLUSH
    opOBJECT_WAIT,LC0(3),                                            //      WAIT(3)
                                                                     //
    opUI_WRITE,LC0(PUT_STRING),LCS,'D','O','N','E','(','3',')','\r','\n',0,  //      UI_WRITE(STRING,"DONE(3)\r\n")
                                                                     //
    opUI_WRITE,LC0(PUT_STRING),LCS,'W','A','I','T','(','2',')','\r','\n',0,  //      UI_WRITE(STRING,"WAIT(3)\r\n")
    opUI_FLUSH,                                                      //      UI_FLUSH
    opOBJECT_WAIT,LC0(3),                                            //      WAIT(2)
                                                                     //
    opUI_WRITE,LC0(PUT_STRING),LCS,'D','O','N','E','(','2',')','\r','\n',0,  //      UI_WRITE(STRING,"DONE(2)\r\n")
     opUI_WRITE,LC0(PUT_STRING),LCS,'D','O','N','E','(','1',')','\r','\n',0,  //      UI_WRITE(STRING,"DONE(1)\r\n")
    opUI_FLUSH,                                                      //      UI_FLUSH
    opOBJECT_END,                                                    //      }
                                                                     //
                                                                     //      BLOCK2
```

```
                                                                    //    {
    opUI_WRITE,LC0(PUT_STRING),LCS,'B','L','O','C','K','2','\r','\n',0,    //    UI_WRITE(STRING,"BLOCK2\r\n")
    opUI_FLUSH,                                                     //    UI_FLUSH
                                                                    //
    opUI_WRITE,LC0(PUT_STRING),LCS,'D','E','L','A','Y','(','2',')','\r','\n',0,//    UI_WRITE(STRING,"DELAY(2)\r\n")
    opUI_FLUSH,                                                     //    UI_FLUSH
    opTIMER_WAIT,LC2(2000),LV0(0),                                  //    DELAY(2)
    opTIMER_READY,LV0(0),                                           //
                                                                    //
    opUI_WRITE,LC0(PUT_STRING),LCS,'T','R','I','G','(','3',')','\r','\n',0,    //    UI_WRITE(STRING,"TRIG(3)\r\n")
    opUI_FLUSH,                                                     //    UI_FLUSH
    opOBJECT_TRIG,LC0(3),                                           //    TRIG(3)
                                                                    //
    opUI_WRITE,LC0(PUT_STRING),LCS,'D','E','L','A','Y','(','6',')','\r','\n',0,//    UI_WRITE(STRING,"DELAY(6)\r\n")
    opUI_FLUSH,                                                     //    UI_FLUSH
    opTIMER_WAIT,LC2(6000),LV0(0),                                  //    DELAY(2)
    opTIMER_READY,LV0(0),                                           //
    opOBJECT_END,                                                   //    }
                                                                    //
                                                                    //    BLOCK3
                                                                    //    {
    opUI_WRITE,LC0(PUT_STRING),LCS,'B','L','O','C','K','3','\r','\n',0,    //    UI_WRITE(STRING,"BLOCK3\r\n")
    opUI_FLUSH,                                                     //    UI_FLUSH
                                                                    //
     opUI_WRITE,LC0(PUT_STRING),LCS,'D','E','L','A','Y','(','2',')','\r','\n',0,//    UI_WRITE(STRING,"DELAY(2)\r\n")
    opUI_FLUSH,                                                     //    UI_FLUSH
     opTIMER_WAIT,LC2(2000),LV0(0),                                 //    DELAY(2)
    opTIMER_READY,LV0(0),                                           //
    opOBJECT_END                                                    //    }
                                                                    //
};
```

7.3 Parallel loops

Program that includes two parallel loops.

```
//  (VMTHREAD1--WRITE--START(2)----------WRITE(One)--DELAY(1)--LOOP10)-----)
//                                \                    ^                    |
//                                 \                   |                    |
//                                  \                  `--------------------'
//                                   (VMTHREAD2--WRITE(One)--DELAY(1)--LOOP10)-----)
//                                                      ^                    |
//                                                      |                    |
//                                                      `--------------------'
UBYTE     prg[] =                                       // p12.c
{                                                       //
    PROGRAMHeader(0,2,0),                               // VersionInfo,NumberOfObjects,GlobalBytes
    VMTHREADHeader(40,5),                               // OffsetToInstructions,LocalBytes
    VMTHREADHeader(106,5),                              // OffsetToInstructions,LocalBytes
                                                        //
                                                        // VMTHREAD1
                                                        // {
    opUI_WRITE,LC0(PUT_STRING),LCS,' ',' ',' ',' ','P', //     UI_WRITE(STRING,"Parallel programming\r\n");
    'a','r','a','l','l','e','l',' ','p','r','o',       //
    'g','r','a','m','m','i','n','g','\r','\n',0,       //
    opUI_FLUSH,                                         //     UI_FLUSH
    opOBJECT_START,LC0(2),                              //     START_OBJECT(2)
                                                        //
    opMOVE8_8,LC0(10),LV0(4),                           //     LV4 = 10
                                                        //     do
                                                        //     {
    opUI_WRITE,LC0(PUT_STRING),LCS,' ',' ',' ',' ','O', //       UI_WRITE(STRING,"    One\r\n");
    'n','e','\r','\n',0,                                //
    opUI_FLUSH,                                         //       UI_FLUSH
    opTIMER_WAIT,LC2(1000),LV0(0),                      //       DELAY(1)
    opTIMER_READY,LV0(0),                               //
                                                        //     }
    opSUB8,LV0(4),LC0(1),LV0(4),                        //     while(--LV4)
    opJR_NEQ8,LV0(4),LC0(0),LC0(-29),                   //
    opOBJECT_END,                                       // }
                                                        //
                                                        // VMTHREAD2
                                                        // {
    opMOVE8_8,LC0(10),LV0(4),                           //   LV4 = 10
                                                        //   do
                                                        //   {
    opUI_WRITE,LC0(PUT_STRING),LCS,' ',' ',' ',' ','T', //     UI_WRITE(STRING,"    Two\r\n");
   'w','o','\r','\n',0,                                 //
    opUI_FLUSH,                                         //     UI_FLUSH
    opTIMER_WAIT,LC2(1000),LV0(0),                      //     DELAY(1)
    opTIMER_READY,LV0(0),                               //
                                                        //   }
    opSUB8,LV0(4),LC0(1),LV0(4),                        //   while(--LV4)
    opJR_NEQ8,LV0(4),LC0(0),LC0(-29),                   //
    opOBJECT_END,                                       // }
                                                        //
};
```

7.4 Calling subroutines with parameters

Below program examples demonstrates how to call subroutines with parameters.

```
UBYTE     prg[] =                                  //  p11
{                                                  //
  PROGRAMHeader(0,2,2),                            //     VersionInfo,NumberOfObjects,GlobalBytes
  VMTHREADHeader(0,2),                             //     OffsetToInstructions,LocalBytes
  SUBCALLHeader(0,2),                              //     OffsetToInstructions,LocalBytes
                                                   //
                                                   //     VMTHREAD1
                                                   //     {
  opMOVE8__8,LC0(1),LV0(0),                        //       LV0  =  1
  opCALL,LC0(2),LC0(2),LV0(0),LV0(1),              //       SUBCALL(2,2,LV0,LV1)
  opUI__WRITE,LC0(VALUE8),LV0(1),                  //       UI__WRITE(VALUE8,LV1)
  opUI__WRITE,LC0(PUT__STRING),LCS,'\r',//       UI__WRITE(PUT__STRING,"\r\n")
  opUI__FLUSH,                                     //       UI__FLUSH
  opOBJECT__END,                                   //     }
                                                   //
  2,IN__8,OUT__8,                                  //     SUBCALL2(InDATA8,OutDATA8)
                                                   //     {
  opUI__WRITE,LC0(VALUE8),LV0(0),                  //       UI__WRITE(VALUE8,LV0)
  opUI__WRITE,LC0(PUT__STRING),LCS,'\r',//       UI__WRITE(PUT__STRING,"\r\n")
  opUI__FLUSH,                                     //       UI__FLUSH
  opMOVE8__8,LV0(0),LV0(1),                        //       LV1  =  2
  opRETURN,                                        //     }
  opOBJECT__END,                                   //
};
```

7.5 String parameter passing

This program shows how to initiate string variable, how to use some string byte codes and pass string parameters to SUBCALL'S.

```
// ALLOCATION ***************************************************************************************
                                                                    //
    enum                                                            // Object enummeration
    {                                                               //
      MAIN__ = 1,                                                   //
      MYSUB__,                                                      //
     OBJECTS,                                                       //
    };                                                              //
                                                                    //
                                                                    //
    #define   GLOBALS           0                                   // Manually sum of globals beneath
                                                                    //
    #define   MAIN__LOCALS      11                                  // Manually sum of locals beneath
    #define   String1           LV1(0)                              // String 1 size is 5  characters + zero termination
    #define   String2           LV1(6)                              // String 2 is an array handle
    #define   String3           LV1(8)                              // String 3 is an array handle
    #define   Dummy             LV1(10)                             // Next variable
                                                                    //
    #define   MYSUB__LOCALS     27                                  // manually sum of locals beneath
    #define   In1               LV1(0)                              // String In1 size is 21 characters + zero
termination
    #define   In2               LV1(22)                             // String In2 is an array handle
    #define   Out               LV1(24)                             // String Out is an array handle
    #define   Result            LV1(26)                             // Result for compare
                                                                    //
    UBYTE     prg[] =                                               // str.c
    {                                                               //
// HEADER *******************************************************************************************
                                                                    //
    PROGRAMHeader(0,(OBJECTS - 1),GLOBALS),                         // VersionInfo,NumberOfObjects,GlobalBytes
    VMTHREADHeader(0,MAIN__LOCALS),                                 // MAIN
    SUBCALLHeader(0,MYSUB__LOCALS),                                 // MYSUB
                                                                    //
// MAIN *********************************************************************************************
                                                                    //
                                                                    // VMTHREAD
                                                                    // {
// opSTRINGS,LC0(DUPLICATE),LCS,'L','E','G','O',0,String1,          //   STRING(DUPLICATE,"LEGO",String1)

    opINIT__BYTES,String1,LC0(5),LC1('L'),LC1('E'),LC1('G'),LC1('O'),LC0(0), //
    opARRAY,LC0(CREATE8),LC0(22),String2,                           //   ARRAY(CREATE8,22,String2)
    opARRAY,LC0(CREATE8),LC0(22),String3,                           //   ARRAY(CREATE8,22,String2)

    opARRAY,LC0(INIT8),String2,LC0(0),LC0(11),LC1('M'),
    LC1('i'),LC1('n'),LC1('d'),LC1('s'),LC1('t'),
    LC1('o'),LC1('r'),LC1('m'),LC1('s'),LC0(0),

//  opINIT__BYTES,HND(String2),LC0(11),LC1('M'),LC1('i'),LC1('n'),LC1('d'),LC1('s'),LC1('t'),
//  LC1('o'),LC1('r'),LC1('m'),LC1('s'),LC0(0),
                                                                    //
//  opSTRINGS,LC0(DUPLICATE),LCS,'M','i','n','d','s','t','o',      //   STRING(DUPLICATE,"Mindstorms",String2)
```

```
//    'r','m','s',0,HND(String2),                          //
      opCALL,LC0(MYSUB__),LC0(3),String1,String2,String3,          //    MYSUB(3,String1,String2,String3)
                                                                   //
      opUI__DRAW,LC0(FILLWINDOW),LC0(0),LC0(0),LC0(0),             //
      opUI__DRAW,LC0(FILLWINDOW),LC0(0),LC0(0),LC0(0),
      opUI__DRAW,LC0(TEXT),LC0(1),LC0(24),LC1(50),HND(String3),//    UI__DRAW(TEXT,1,24,50,String3)
      opUI__DRAW,LC0(UPDATE),                                      //    UI__DRAW(UPDATE)
      opOBJECT__END,                                               //  }
                                                                   //
// MYSUB SUBCALL ***************************************************************************************
                                                                   //
      0x03,IN__S,(22),IN__16,IO__16,                               //  MYSUB(In1,In2,Out)
                                                                    //  {
      opSTRINGS,LC0(COMPARE),In1,LCS,'L','E','G','O',0,Result,     //    STRING(COMPARE,In1,"LEGO",Result)
      opJR__FALSE,Result,LC0(12),                                  //    if (Result != 0)
                                                                   //    {
      opSTRINGS,LC0(ADD),In1,LCS,' ',0,In1,                        //      STRING(ADD,In1," ",In1)
      opSTRINGS,LC0(ADD),In1,HND(In2),HND(Out),                    //      STRING(ADD,In1,In2,Out)
                                                                   //    }
      opRETURN,                                                    //  }
      opOBJECT__END,                                               //
                                                                   //
// END *************************************************************************************************
```