LEGO MindStorms

RCX 2.0 Firmware

Command Overview

BYTE CODE COMMAND INTERPRETER 5

VIRTUAL MACHINE CHARACTERISTICS 5
BYTE CODE COMMAND STRUCTURE 5
PARAMETER SOURCES 6
BYTE CODE COMMANDS 11
PBAliveOrNot (0x10) *12*
MemMap (0x20) *13*
PBBattery (0x30) *14*
DeleteAllTasks (0x40) *15*
StopAllTasks (0x50) *16*
PBTurnOff (0x60) *17*
DeleteAllSubs (0x70) *18*
ClearSound (0x80) *19*
ClearPBMessage (0x90) *20*
ExitAccessControl (0xA0) *21*
ExitEventCheck (0xB0) *22*
MuteSound (0xD0) *23*
UnmuteSound (0xE0) *24*
ClearAllEvents (0x06) *25*
EndOfSub (0xF6) *26*
OnOffFloat (0x21) *27*
PbTXPower (0x31) *28*
PlaySystemSound (0x51) *29*
DeleteTask (0x61) *30*
StartTask (0x71) *31*
StopTask (0x81) *32*
SelectProgram (0x91) *33*
ClearTimer (0xA1) *34*
PBPowerDownTime (0xB1) *35*
DeleteSub (0xC1) *36*
ClearSensorValue (0xD1) *37*
SetFwdSetRwdRewDir (0xE1) *38*
Gosub (0x17) *39*
SJump (0x27) *40*
SCheckLoopCounter (0x37) *41*
ConnectDisconnect (0x67) *42*
SetNormSetInvAltDir (0x77) *43*
IncCounter (0x97) *44*
DecCounter (0xA7) *45*
ClearCounter (0xB7) *46*
SetPriority (0xD7) *47*
InternMessage (0xF7) *48*
PlayToneVar (0x02) *49*
Poll (0x12) *50*
SetWatch (0x22) *51*
SetSensorType (0x32) *52*
SetSensorMode (0x42) *53*
SetDataLog (0x52) *54*
DataLogNext (0x62) *55*

LJump (0x72) *56*
SetLoopCounter (0x82) *57*
LCheckLoopCounter (0x92) *58*
SendPBMessage (0xB2) *59*
SendUARTData (0xC2) *60*
RemoteCommand (0xD2) *61*
SDecVarJumpLTZero (0xF2) *62*
DirectEvent (0x03) *63*
SetPower (0x13) *64*
PlayTone (0x23) *65*
SelectDisplay (0x33) *66*
Wait (0x43) *67*
UploadRam (0x63) *68*
EnterAccessControl (0x73) *69*
SetEvent (0x93) *70*
SetMaxPower (0xA3) *72*
LDecVarJumpLTZero (0xF3) *73*
CalibrateEvent (0x04) *74*
SetVar (0x14) *75*
SumVar (0x24) *76*
SubVar (0x34) *77*
DivVar (0x44) *78*
MulVar (0x54) *79*
SgnVar (0x64) *80*
AbsVar (0x74) *81*
AndVar (0x84) *82*
OrVar (0x94) *83*
Upload (0xA4) *84*
SEnterEventCheck (0xB4) *85*
SetSourceValue (0x05) *86*
UnlockPBrick (0x15) *87*
BeginOfTask (0x25) *88*
BeginOfSub (0x35) *89*
ContinueFirmwareDownLoad, ContinueDL (0x45) *90*
GoIntoBootMode (0x65) *91*
BeginFirmwareDownLoad (0x75) *92*
SCheckDo (0x85) *93*
LCheckDo (0x95) *94*
UnlockFirmware (0xA5) *95*
LEnterEventCheck (0xB5) *96*
ViewSourceValue (0xE5) *97*

MOTORS **98**

EVENTS **100**

VALID COMBINATIONS 101
EVENT GENERATION 101
THRESHOLDS AND HYSTERESIS 103
AUTOMATIC CALIBRATION 103
Temperature mode *104*

Reflection mode *104*
Raw mode *104*
All other modes *105*

SHARED RESOURCES **106**

ACCESS CONTROL & EVENT MONITORING **107**

Byte Code Command Interpreter

The command interpreter executes low level byte code commands – either from downloaded programs or from direct commands received from either a controlling computer, a remote control or from another P-brick. The interpretation is based on a virtual machine.

Virtual machine characteristics

The module implements a virtual machine with the following characteristics:

- 5 program slots, each containing 10 tasks and 8 subroutines
- 32 persistent global variables, shared between all tasks and subroutines. Persistent means that the variables are only initialized to zero during power-up as e.g. after changing batteries
- 16 local variables for each task that can be used for 'safe' parameter passing to subroutines. Local variables are only available to the tasks and subroutines in a running program.
- 4 system timers, shared between all tasks and subroutines. The timers continuously run from 0 to 32767 (0x7FFF), where the timers wrap around (starts from 0 again). The timers have a 100 ms resolution.
- 3 input ports
- 3 output port
- 1 sound unit
- 1 LCD display unit

Byte code command structure

The virtual machine executes byte commands following the general pattern:

Opcode byte	Parameter bytes…
X X X X T N N N	...

The 'T' field is for flow control (a 'toggle' bit) – re-sent commands keep the bit whereas new commands will toggle it to indicate that something new was send and that it should be executed.

The combination 'X X X X N N N' is the actual byte code.

The 'N N N' (the three lowest bits – 3 LSB) tells how many parameter bytes to expect after the opcode[1]. This is useful if commands intended for another type of P-brick was accidentally sent – if the command is not recognised, the interpreter can skip past it.

[1] For program size efficiency, the (unused) lenghts 6 and 7 actually means 0 and 1 respectively. It doubles the number of available commands with few parameters.

Parameter sources

Many commands accept parameters in the form of a source and value pair. The source tells the type of the parameter and the value is typically the index of that particular type of parameter.

The currently available sources are:

Source	Name	Range	Access	Comments
0	Variable	I: 0-31 D:0-47	Read-Write	Variables are 16-bit signed values. Value: -32768 - 32767
1	Timer	0-3	Read-Write	Timers are 15-bit unsigned (actually 16-bit signed with 0x7FFF-0 wrap-around) with 100 ms resolution. Writing to them also affects source 26 high-resolution timers and vice-versa. Value: 0 - 32767
2	Constant	-32768-32767 0-0xFFFF	Read-only	Different commands accept different actual values
3	Motor status	0-2	Read-only	The values are 8-bit wide fields, formatted as: bit 0-2 == power setting bit 3-3 == direction (1 == FWD, 0 == RWD) bit 4-5 == remote command overlay marker bit 6-6 == brake/float (1 == brake, 0 == float) bit 7-7 == on/off (1 == on, 0 == off) when bit 7 is set (the motor is on), bit 6 is ignored
4	Random	R: 1-32767 R: 1-0x7FFF W: 0	Read-Write	The source generates a random number between 0 and the actual given range. Writing to random 0 seeds the random number generator.
8	Program slot	0	Read-Write	Returned and accepted values are in the 0-4 range. Writing changes the selected program slot (and starts it, if under program or remote control).
9	Sensor value	0-2	Read-only	The returned value is generated in the firmware based on sensor type and mode

Source	Name	Range	Access	Comments
10	Sensor type	0-2	Read-Write	The values are 8-bit numbers, meaning: 0 == NoSensor 1 == Switch 2 == Temperature 3 == Reflection 4 == Angle
11	Sensor mode	0-2	Read-Write	The values are 8-bit fields, with the bits meaning: Bit 0-4: 0x00 == Absolute measurement 0x01 … 0x1F == Dynamic measurement Bit 5-7: 0x00 == RawMode 0x20 == BooleanMode 0x40 == TransitionCntMode 0x60 == PeriodCounterMode 0x80 == PctFullScaleMode 0xA0 == CelsiusMode 0xC0 == FahrenheitMode 0xE0 == AngleStepsMode
12	Sensor raw	0-2	Read-only	The un-processed 10 bit A/D value, straight off the hardware. Value: 0 - 1023
13	Sensor bool	0-2	Read-only	The raw value, with thresholds and hysteresis at 45/55%. Value: 0 - 1
14	Watch	0	Read-Write	The source is interpreted as “hours*60 + minutes” and cannot exceed 23 hrs and 59 min.
15	Message	0	Read-Write	The value is an 8-bit unsigned number received from another P-brick. When writing to the source, only the lower 8 bit of the value are used.

Source	Name	Range	Access	Comments
17	Global motor status	0-2	Read-only	The values are 8-bit wide fields, formatted as: bit 0-2 == max power setting bit 3-3 == global direction (1 == Normal, 0 == Reverse) bit 4-5 == remote command overlay marker bit 7-7 == global on/off (1 == on, 0 == off)
21	Counter	0-2	Read-Write	The counters are identical to global variables 0-2. The difference is that they can be used to generate events based on thresholds. Value: -32768 - 32767
23	Task events	0-9	Read-only	Contains a copy of the bit register of the event(s) that lead to event monitoring succeeding
25	Event state	0-15	Read-only	Tells whether the event is in the low, normal or high state depending on the set thresholds. The event state may also be undefined or calibrating. 0 == Low 1 == Normal 2 == High 3 == Undefined 4 == Start calibrating 5 == Calibrating in process
26	10 ms timer	0-3	Read-Write	Timers are 15-bit unsigned (actually 16-bit signed with 0x7FFF-0 wrap-around). 10 ms resolution. These timers are derived from source 1 and only generated on demand. Writing to them also affects source 1 timers and vice-versa. Value: 0 - 32767
27	Click counter	0-15	Read-Write	Contains the number of Clicks detected by firmware for the event sensor. Value: -32768 - 32767

Source	Name	Range	Access	Comments
28	Upper threshold	0-15	Read-Write	Used for event generation. The actual values should be in native units (source 0, 1, 9, 15 and 21). When specifying temperature thresholds the threshold must be given as temperature multiplied by 10. Value: -32768 - 32767
29	Lower threshold	0-15	Read-Write	As above.
30	Hysteresis	0-15	Read-Write	As above. The value must be non-negative. Value: 0 - 32767
31	Duration	0-15	Read-Write	The value is measured in 10 ms units. Value: 5 - 32767 (50ms - ~327s)
33	UART setup	0-15: Data 16-17: Setup	Read-Write	The values are 8-bit values [0-15] Data to be sent using the setup below [16] PackConf – data formatting: bit 0 : Preamble (0x55 0xFF 0x00) bit 1 : Negated (incl. negated checksum) bit 2 : Checksum (if not negated) [17] UartConf – transmit parameters: bit 0 : Baudrate (0 == 2400 baud, 1 == 4800 baud) bit 1 : Carrier freq (0 == 38 kHz, 1 == 76 kHz) bit 2 : Dutycycle (0 == 50 %, 1 == 25 %)
34	Battery level	0	Read-only	The value is the battery voltage level multiplied by 1000.
35	Firmware version	0	Read-only	The value is the firmware version number multiplied by 100.

Source	Name	Range	Access	Comments
36	Indirect variable	I: 0-31 D:0-47	Read-Write	This source first looks up the indicated (index/pointer) variable and then uses the value of that variable to look up the final variable which is either read from or written to. It enables array-based access to variables and efficient coding of program loop bodies. One can (indirectly) read a value into a work variable, perform the processing and the write (indirectly) the value back, before incrementing or decrementing the (index/pointer) variable.

Strictly speaking, there are no write-only sources, but some commands pack information together in registers, which can later be accessed as read-only and unpacked by a program.

Byte code commands

The following sections describe all the byte code commands in terms of command layout, expected return answer from the RCX2, LASM syntax and an explanation of what the command does.

In this overview, all command and reply byte codes have the transmission toggle bit set to zero.

There is an important different between LASM commands and the actual byte codes themselves when it comes to relative jump distances. When given as absolute numbers, LASM counts from the **start** of the command but the actual value in the (jump) distance field is **relative to the position of the field itself**. This means that an infinite loop is written in LASM as "`jmp 0`" but the generated byte codes are "0x27 0x81" i.e. jump 1 byte backwards.

Please refer to the source/value table on the preceding pages for additional information when reading about the individual commands.

PBAliveOrNot (0x10)

Command structure:

0x10

Reply:

0xE7

Availability:

Direct/Program

LASM syntax:

```
ping
```

Explanation:

The command tests for the presence of a P-brick and re-initialises the toggle bit from the communication protocol in the RCX, when executed as a direct command. When executed as a program command, it resets the power-down timer.

MemMap (0x20)

Command structure:

0x20

Reply:

0xD7	HI(Sub00)	LO(Sub00)	…	HI(Sub07)	LO(Sub07)			
	…	…	…	…	…			
	HI(Sub40)	LO(Sub40)	…	HI(Sub47)	LO(Sub47)			
	HI(Task00)	LO(Task00)	…				HI(Task09)	LO(Task09)
	…	…	…				…	…
	HI(Task40)	LO(Task40)	…				HI(Task49)	LO(Task49)
	HI(Log)	LO(Log)						
	HI(CurrentLog)	LO(Currentlog)						
	HI(MemUsed)	LO(MemUsed)						
	HI(MemTop)	LO(MemTop)						

Availability:

Direct

LASM syntax:

```
memmap
```

Explanation:

The command returns the addresses for the downloaded program tasks, subroutines and data log area. They can then be used to calculate the remaining memory size or in subsequent calls to UploadRam.

PBBattery (0x30)

Command structure:

0x30

Reply:

0xC7	Value (LO)	Value (HI)

Availability:

Direct

LASM syntax:

```
pollb
```

Explanation:

The command returns the current battery level multiplied by 1000.

DeleteAllTasks (0x40)

Command structure:

0x40

Reply:

0xB7

Availability:

Direct

LASM syntax:

```
delt
```

Explanation:

The command deletes all the tasks in the currently selected program.

Before being deleted, the tasks are stopped and their acquired access resources released.

StopAllTasks (0x50)

Command structure:

0x50

Reply:

0xA7

Availability:

Direct/Program

LASM syntax:

```
stop
```

Explanation:

The command stops all (running) tasks in the currently selected program and releases acquired access resources.

PBTurnOff (0x60)

Command structure:

0x60

Reply:

0x97

Availability:

Direct/Program

LASM syntax:

```
offp
```

Explanation:

The command turns the RCX off by resetting the power-down timer to 0.

DeleteAllSubs (0x70)

Command structure:

0x70

Reply:

0x87

Availability:

Direct

LASM syntax:

```
dels
```

Explanation:

The command deletes all subroutines in the currently selected program.

Tasks that call subroutines are stopped as a precaution. Access resources acquired by those tasks are released.

ClearSound (0x80)

Command structure:

0x80

Reply:

0x77

Availability:

Direct/Program

LASM syntax:

```
playz
```

Explanation:

The command immediately empties the sound buffer in the RCX from any and all queued tones or system sounds. This can be used to implement immediate sound feedback for events.

ClearPBMessage (0x90)

Command structure:

0x90

Reply:

0x67

Availability:

Direct/Program

LASM syntax:

```
msgz
```

Explanation:

The command resets the IR message buffer to 0.

ExitAccessControl (0xA0)

Command structure:

0xA0

Availability:

Program

LASM syntax:

```
monax
```

Explanation:

The command exits the access control region if the task were inside one. It is assumed that the task leaves the resources in the desired state, as the command will not change their setup.

ExitEventCheck (0xB0)

Command structure:

0xB0

Availability:

Program

LASM syntax:

```
monex
```

Explanation:

The command stops event monitoring for the task, if it were running any.

MuteSound (0xD0)

Command structure:

0xD0

Reply:

0x27

Availability:

Direct/Program

LASM syntax:

```
mute
```

Explanation:

The command empties the sound buffer and ignores future sounds requested by the command interpreter. System generated sounds from e.g. button presses or low battery detection are still processed.

UnmuteSound (0xE0)

Command structure:

0xE0

Reply:

0x17

Availability:

Direct/Program

LASM syntax:

```
speak
```

Explanation:

The command restarts the processing of sounds requested by the command interpreter.

ClearAllEvents (0x06)

Command structure:

0x06

Reply:

0xF1

Availability:

Direct/Program

LASM syntax:

```
dele
```

Explanation:

The command clears all the 16 events. Tasks that are actively waiting/monitoring for events are **not** alerted to this fact.

EndOfSub (0xF6)

Command structure:

0xF6

Availability:

Program

LASM syntax:

```
rets
```

Explanation:

The command is inserted at the end of every downloaded subroutine, but may be used as a (subroutine) program command to return at the earliest possible moment, maybe to get an easier-to-understand flow-control.

The command checks to see if task has any event monitoring or access control region active inside the subroutine and exits them if necessary (releasing the resources).

After returning, the subroutine return address is cleared.

If called outside of a subroutine (when the return address is cleared) the firmware command interpreter will stop the task as something has gone completely wrong. Access resources acquired by the task will be released.

OnOffFloat (0x21)

Command structure:

0x21	Bit 0-2: motor list Bit 6-7: float (0), off (1), on (2)

Reply:

0xD6

Availability:

Direct/Program

LASM syntax:

```
out onofffloat, motor list
```

Explanation:

The command sets the Motor Status register with the given values.

PbTXPower (0x31)

Command structure:

0x31	0: short range 1: long range

Reply:

0xC6

Availability:

Direct/Program

LASM syntax:

```
txs range
```

Explanation:

The command sets the IR transmission range in the RCX.

PlaySystemSound (0x51)

Command structure:

0x51	0: key click sound 1: beep sound 2: sweep up 3: sweep down 4: error sound 5: fast sweep up

Reply:

0xA6

Availability:

Direct/Program

LASM syntax:

```
plays sound
```

Explanation:

The command plays the given system sound, if sounds are globally enabled.

DeleteTask (0x61)

Command structure:

0x61	Task number (0-9)

Reply:

0x96

Availability:

Direct

LASM syntax:

```
delt task number
```

Explanation:

The command deletes a task. If the task was running, it is stopped and its access resources released.

StartTask (0x71)

Command structure:

0x71	Task number (0-9)

Reply:

0x86

Availability:

Direct/Program

LASM syntax:

```
start task number
```

Explanation:

The command (re)starts the given task if the task slot is non-empty.

StopTask (0x81)

Command structure:

0x81	Task number (0-9)

Reply:

0x76

Availability:

Direct/Program

LASM syntax:

```
stop task number
```

Explanation:

The command stops the given task and releases its access resources.

SelectProgram (0x91)

Command structure:

0x91	Program number (0-4)

Reply:

0x66

Availability:

Direct/Program

LASM syntax:

```
prgm program slot
```

Explanation:

The command stops all tasks in the currently selected program and then selects the given program slot.

If the command is executed as part of a downloaded program, task 0 is started in the new program. This allows for chaining together of programs and the functionality was added to be consistent with the behaviour exhibited by the LEGO MindStorms remote control.

The same differentiated behaviour can be observed using the SetSourceValue command to write to source 8.

It furthermore follows a philosophy of providing richer functionality at the programming level than at the button/key level.

ClearTimer (0xA1)

Command structure:

0xA1	Timer number (0-3)

Reply:

0x56

Availability:

Direct/Program

LASM syntax:

```
tmrz timer number
```

Explanation:

The command resets the given system timer to zero.

PBPowerDownTime (0xB1)

Command structure:

0xB1	Power down time in minutes (0 == don't time out)

Reply:

0x46

Availability:

Direct/Program

LASM syntax:

```
tout power down time
```

Explanation:

The command sets the power down time to the given amount and resets the power down timer accordingly. The timer is reset every time a command is received or every time the downloaded program executed the PBAliveOrNot command.

The power down timer is controlled by a "low level" timer, counting in seconds. This "low level" timer can not be reset, therefore an inaccuracy on the power down timer must be expected.

DeleteSub (0xC1)

Command structure:

0xC1	Subroutine number (0-7)

Reply:

0x36

Availability:

Direct

LASM syntax:

```
dels subroutine number
```

Explanation:

The command deletes the given subroutine and stops all tasks that could be calling this subroutine. Access resources acquired by those tasks are released.

ClearSensorValue (0xD1)

Command structure:

0xD1	Sensor number (0-2)

Reply:

0x26

Availability:

Direct/Program

LASM syntax:

```
senz sensor number
```

Explanation:

The command resets the source 9 register for the sensor. The current sensor value is computed again as part of the normal sensor processing.

This is only really useful for resetting an angle sensor (that counts rotations in 1/16 ticks).

SetFwdSetRwdRewDir (0xE1)

Command structure:

0xE1	0-2: motor list 6-7: backwards (0), reverse (1), forwards (2)

Reply:

0x16

Availability:

Direct/Program

LASM syntax:

```
dir direction, motor list
```

Explanation:

The command sets the Motor Status register with the given values.

Gosub (0x17)

Command structure:

0x17	Subroutine number (0-7)

Availability:

Program

LASM syntax:

```
calls subroutine number
```

Explanation:

The command moves the tasks instruction pointer to the start of the given subroutine and saves the address of the next instruction in the task as the subroutine return address.

This command should only ever be called from within a task. Executing it from within another subroutine will corrupt the return address for the calling subroutine eventually causing the task that originally called the offending subroutine to be stopped.

SJump (0x27)

Command structure:

0x27	0-6: jump distance 7-7: forwards (0), backwards (1)

Availability:

Program

LASM syntax:

```
jmp label or offset from start of command
```

Explanation:

The command jumps a certain distance, either forwards or backwards.

An infinite loop can be implemented using the command '**jmp** 0'.

SCheckLoopCounter (0x37)

Command structure:

0x37	Jump distance

Availability:

Program

LASM syntax:

```
loopc forward label or forward distance
```

Explanation:

The command checks the current loop counter. If it is 0 (zero), the command removes the current (nested) loop level and jumps forward, otherwise the current loop counter is decremented.

If there is no current or active loop level, the command is ignored.

It is **recommended not to use this command at all** since it relies on internal state information which may become corrupted if the program flow-of-control changes due to firmware generated events or access control revocations. **Instead use the DecVarJumpLTZero commands** as they use ordinary variables that can be manipulated in many other ways for controlled loop termination.

ConnectDisconnect (0x67)

Command structure:

0x67	Bit 0-2: motor list Bit 6-7: float (0), off (1), on (2)

Reply:

0x90

Availability:

Direct/Program

LASM syntax:

```
gout onofffloat, motor list
```

Explanation:

The command sets the Global Motor Status register with the given values.

SetNormSetInvAltDir (0x77)

Command structure:

0x77	0-2: motor list 6-7: backwards (0), reverse (1), forwards (2)

Reply:

0x80

Availability:

Direct/Program

LASM syntax:

```
gdir direction, motor list
```

Explanation:

The command sets the Global Motor Status register with the given values.

IncCounter (0x97)

Command structure:

0x97	Counter number (0-2)

Reply:

0x60

Availability:

Direct/Program

LASM syntax:

```
cnti counter number
```

Explanation:

The command increments the given counter (which equals the global variable with the same number) and then checks for possible events.

DecCounter (0xA7)

Command structure:

0xA7	Counter number (0-2)

Reply:

0x50

Availability:

Direct/Program

LASM syntax:

```
cntd counter number
```

Explanation:

The command decrements the given counter (which equals the global variable with the same number) and then checks for possible events.

ClearCounter (0xB7)

Command structure:

0xB7	Counter number (0-2)

Reply:

0x40

Availability:

Direct/Program

LASM syntax:

```
cntz counter number
```

Explanation:

The command resets the given counter (which equals the global variable with the same number) and then checks for possible events.

SetPriority (0xD7)

Command structure:

0xD7	Task priority

Availability:

Program

LASM syntax:

```
setp priority
```

Explanation:

The command sets the priority level for the given tasks. All 256 levels can be used, with level 0 having highest priority.

InternMessage (0xF7)

Command structure:

0xF7	IR message

Reply:

There is no reply to this command.

Availability:

Direct/Program

LASM syntax:

```
msgs non-zero message
```

Explanation:

The command sets the IR message buffer with the given value and then checks for possible events.

PlayToneVar (0x02)

Command structure:

0x02	Variable number	Duration in 1/100 sec

Reply:

0xF5

Availability:

Direct/Program

LASM syntax:

```
playv variable number, duration
```

Explanation:

Gets the variable contents as the frequency and puts the tone and duration in the RCX sound buffer.

Poll (0x12)

Command structure:

0x12	Poll source	Poll value

Reply:

0xE5	Value (LO)	Value (HI)

Availability:

Direct

LASM syntax:

```
poll source, value
```

Explanation:

Reads the given source and value combination and sends the reply back.

Task local variables cannot be polled.

SetWatch (0x22)

Command structure:

0x22	Hours	Minutes

Reply:

0xD5

Availability:

Direct/Program

LASM syntax:

```
setw hours, minutes
```

Explanation:

Sets the internal watch to the given time.

The Watch will be selected and the new time will be shown in the LCD display when this is updated.

SetSensorType (0x32)

Command structure:

0x32	Sensor number	Sensor type: 0 == NoSensor 1 == Switch 2 == Temperature 3 == Reflection 4 == Angle

Reply:

0xC5

Availability:

Direct/Program

LASM syntax:

```
sent sensor number, sensor type
```

Explanation:

Sets the given sensor to the given type. Resets previous sensor state and sensor information. Default sensor modes are assigned or each type of sensor.

- NoSensor => RawMode
- Switch => BooleanMode
- Temperature => CelsiusMode
- Reflection => PctFullScaleMode
- Angle => AngleStepsMode

For different modes, use the SetSensorMode command **afterwards**.

SetSensorMode (0x42)

Command structure:

0x42	Sensor number	Bit 0-4 (slope): 0 == absolute measurement 1-31 == dynamic measurement (slope) Bit 5-7 (mode): 0 == RawMode 1 == BooleanMode 2 == TransitionCntMode 3 == PeriodCounterMode 4 == PctFullScaleMode 5 == CelsiusMode 6 == FahrenheitMode 7 == AngleStepsMode

Reply:

0xB5

Availability:

Direct/Program

LASM syntax:

```
senm sensor number, sensor mode, slope
```

Explanation:

Sets the given mode and measurement slope for the given sensor.

The command also resets the processed sensor value.

SetDataLog (0x52)

Command structure:

0x52	Datalog size (LO)	Datalog size (HI)

Reply:

0xA5

Availability:

Direct/Program

LASM syntax:

```
logz datalog size
```

Explanation:

Clears the current data log buffer and allocates room for a new one of the given size.

In fact room is allocated for an extra element to hold the number of data points currently in the log. Since that element is then used, the initial number is set to 1. Its type field as described in the DataLogNext command is set to the value 0xFF to distinguish it from other values.

This means that subsequent “real” data points appear as elements 1 and upwards.

DataLogNext (0x62)

Command structure:

0x62	Datalog source	Datalog value

Reply:

0x95

Availability:

Direct/Program

LASM syntax:

```
log datalog source, datalog value
```

Explanation:

Stores a 3 byte data point in the log. If there is no more space in the log, the data point is ignored.

Data point structure:

Bit 0-4: Source index Bit 5-7: Source type 0 == variable 1 == timer 2 == sensor value 4 == watch	Value (LO)	Value (HI)

Task local variables cannot be logged as their source index lies outside the range that can be represented in the 5 bits from 0-4 (5 bits cover 32 index values).

LJump (0x72)

Command structure:

0x72	Bit 0-6: jump distance (LO) Bit 7: jump direction 0 == forwards 1 == backwards	Jump distance (HI)

Availability:

Program

LASM syntax:

```
jmpl label or offset from start of command
```

Explanation:

The command jumps a certain distance, either forwards or backwards.

An infinite loop can be implemented using the command '**jmpl** 0'.

SetLoopCounter (0x82)

Command structure:

0x82	Loop counter source	Loop counter value

Availability:

Program

LASM syntax:

```
loops loop counter source, loop counter value
```

Explanation:

The command tries to increment the current loop nesting level and set the loop counter for that level to the given value.

There are only 4 (four) loop levels and if they are all in use, the command is ignored.

If the given value lies outside the 1-255 range, the value 0 (zero) is used instead. This will cause a subsequent CheckLoopCounter command to exit immediately.

It is **recommended not to use this command at all** since it relies on internal state information which may become corrupted if the program flow-of-control changes due to firmware generated events or access control revocations. **Instead use the DecVarJumpLTZero commands** as they use ordinary variables that can be manipulated in many other ways for controlled loop termination.

LCheckLoopCounter (0x92)

Command structure:

0x92	Jump distance (LO)	Jump distance (HI)

Availability:

Program

LASM syntax:

```
loopcl forward label or forward distance
```

Explanation:

The command checks the current loop counter. If it is 0 (zero), the command removes the current (nested) loop level and jumps forward, otherwise the current loop counter is decremented.

If there is no current or active loop level, the command is ignored.

It is **recommended not to use this command at all** since it relies on internal state information which may become corrupted if the program flow-of-control changes due to firmware generated events or access control revocations. **Instead use the DecVarJumpLTZero commands** as they use ordinary variables that can be manipulated in many other ways for controlled loop termination.

SendPBMessage (0xB2)

Command structure:

0xB2	Message source	Message value

Availability:

Program

LASM syntax:

```
msg source, value
```

Explanation:

Gets the given value and sends the low 8 bits of that value as an IR message (InternMessage, 0xF7) for other RCXs or Scouts to receive.

SendUARTData (0xC2)

Command structure:

0xC2	Data start	Data size

Reply:

There is no reply to this command.

Availability:

Direct/Program

LASM syntax:

```
uart start, size
```

Explanation:

The command reads the communication setup from the UART source (33) and then sends data from the same buffer accordingly.

The condition “start + size <= 16” must be satisfied to prevent overrun when reading from the buffer.

With proper setup of the data formatting (0x03) and transmit control parameters (0x00), the command **can** be used to send IR commands to other RCXs – just look up the byte codes in this document, store the desired values somewhere in the data buffer and call this command. Look out for the toggle bit **and** the reply from the other P-brick (some commands have replies that can have catastrophic results, so do this at your own risk).

RemoteCommand (0xD2)

Command structure:

0xD2	Remote command (LO) 0x01 == Motor C backwards 0x02 == Program 1 0x04 == Program 2 0x08 == Program 3 0x10 == Program 4 0x20 == Program 5 0x40 == Stop program & motors 0x80 == Remote sound	Remote command (HI) 0x01 == PBMessage 1 0x02 == PBMessage 2 0x04 == PBMessage 3 0x08 == Motor A forwards 0x10 == Motor B forwards 0x20 == Motor C forwards 0x40 == Motor A backwards 0x80 == Motor B backwards

Reply:

There is no reply to this command.

Availability:

Direct

LASM syntax:

```
remote remote command
```

Explanation:

The motor commands are interpreted continuously, the others are one-shot commands.

To execute several remote commands, send a '`remote 0`' between them to clear internal buffers.

SDecVarJumpLTZero (0xF2)

Command structure:

0xF2	Variable number	Bit 0-6: jump distance Bit 7: jump direction 0 == forwards 1 == backwards

Availability:

Program

LASM syntax:

```
decvjn variable number, label or jump distance
```

Explanation:

Decrements the variable and jumps if the variable is negative afterwards.

This is the preferred way of implementing loops as it does not rely on internal state information like the SetLoopCounter/CheckLoopCounter commands. Since it uses ordinary variables it also allows direct access to the loop counter for conditional termination of loops.

DirectEvent (0x03)

Command structure:

0x03	Event source	Event value (LO)	Event value (HI)

Reply:

0xF4

Availability:

Direct/Program

LASM syntax:

```
event event source, event value
```

Explanation:

Forces the firmware to behave as if the events, whose bits are set in the calculated 16 bit value, had actually happened.

SetPower (0x13)

Command structure:

0x13	Motor list	Power source	Power value

Reply:

0xE4

Availability:

Direct/Program

LASM syntax:

```
pwr motor list, power source, power value
```

Explanation:

Sets the Motor Status registers with the new power levels.

PlayTone (0x23)

Command structure:

0x23	Frequency (LO)	Frequency (HI)	Duration

Reply:

0xD4

Availability:

Direct/Program

LASM syntax:

```
playt frequency, duration
```

Explanation:

Queues the tone in the RCX sound buffer, if enabled.

SelectDisplay (0x33)

Command structure:

0x33	View source	View value (LO)	View value (HI)

Reply:

0xC4

Availability:

Direct/Program

LASM syntax:

```
view view source, view value
```

Explanation:

The number is evaluated and the LCD display will then show and update the corresponding data source. The possible values are:

- 0 == Watch
- 1 == Input 1
- 2 == Input 2
- 3 == Input 3
- 4 == Output A
- 5 == Output B
- 6 == Output C
- 7 == User Selection

The User Selection is further described in the ViewSourceValue command.

Wait (0x43)

Command structure:

0x43	Wait source	Wait value (LO)	Wait value (HI)

Availability:

Program

LASM syntax:

```
wait wait source, wait value
```

Explanation:

Suspends the task execution for the given number of 10 ms.

If the given number is negative, the command is ignored.

UploadRam (0x63)

Command structure:

0x63	RAM address (LO)	RAM address (HI)	Byte count

Reply:

0x94	Byte @ address	…	Byte @ (address + byte count – 1)

Availability:

Direct

LASM syntax:

```
pollm start address, number of bytes
```

Explanation:

Uploads raw data from the RCX.

Data can be read from the addresses 0x8000 - 0xFD80.

A maximum of 150 bytes is allowed because of automatic shutdown in the IR tower.

EnterAccessControl (0x73)

Command structure:

0x73	Resources 0x01 == Motor A 0x02 == Motor B 0x04 == Sound 0x08 == Motor C	Bit 0-6: Jump distance (LO) Bit 7: Jump direction 0 == forwards 1 == backwards	Jump distance (HI)

Availability:

Program

LASM syntax:

```
monal resources, label or jump distance
```

Explanation:

Tries to gain access to the requested resources (OR them together to request more than one) with the tasks priority. If the task succeeds, execution continues. If the task fails to acquire the resources or if the resources are later claimed by another task with a higher priority (before releasing the resources), execution continues from the address given in the command.

The slightly strange resource values gives Scout compatibility where the VLL diode mimics a third motor.

The command and underlying state machine accepts 8 different resources (each having a bit in the resource field) so the user can implement priority based access to other (virtual) resources. Since the state machine cannot know about such other resources, the user is responsible for releasing them when access is revoked.

SetEvent (0x93)

Command structure:

0x93	Event number (0-15)	Event sensor number (0-10) 0-2 Input 1-3 (Source 9) 3-6 Timers 0-3 (Source 1) 7 Mailbox (Source 15) 8-10 Counter 0-2 (Source 21)	Event type (0-16) 0 Pressed 1 Released 2 Period 3 Transision 7 > Changerate 8 Enter low 9 Enter normal 10 Enter high 11 Click 12 Double click 14 Mail box 16 Reset event

Reply:

0x64

Availability:

Direct/Program

LASM syntax:

```
sete event number, sensor number, event type
```

Explanation:

Tells the firmware to start looking for the given event type on the given sensor and set the given event bit when the event occurs. An event type of 16 deletes (clears) the event.

Default upper and lower thresholds as well as hysteresis are set depending on the physical sensor mode as presented in the table below:

Physical sensor mode	**Lower threshold**	**Upper threshold**	**Hysteresis**
0 == RawMode	0	1023	0
1 == BooleanMode	0	1	0
2 == TransitionCntMode	0	32767	0
3 == PeriodCounterMode	0	32767	0
4 == PctFullScaleMode	0	100	0
5 == CelsiusMode	-200	700	0
6 == FahrenheitMode	-40	1580	0
7 == AngleStepsMode	0	32767	0

Virtual sensor type			
Timer	0	32767	0
Counter	-32768	32767	0
IR Message	0	255	0

In addition to these values, the default duration time (used for Click event generation) is set to 150 ms and the Click counter is correspondingly cleared (set to zero).

SetMaxPower (0xA3)

Command structure:

0xA3	Motor list	Max power source	Max power value

Reply:

0x54

Availability:

Direct/Program

LASM syntax:

```
gpwr motor list, power source, power value
```

Explanation:

Sets the Global Motor Status registers with the new power levels.

LDecVarJumpLTZero (0xF3)

Command structure:

0xF2	Variable number	Bit 0-6: Jump distance (LO) Bit 7: Jump direction 0 == forwards 1 == backwards	Jump distance (HI)

Availability:

Program

LASM syntax:

```
decvjnl variable number, label or jump distance
```

Explanation:

Decrements the variable and jumps if the variable is negative afterwards.

This is the preferred way of implementing loops as it does not rely on internal state information like the SetLoopCounter/CheckLoopCounter commands. Since it uses ordinary variables it also allows direct access to the loop counter for conditional termination of loops.

CalibrateEvent (0x04)

Command structure:

0x04	Event number	Lower threshold percentage	Upper threshold percentage	Hysteresis percentage

Reply:

0xF3

Availability:

Direct/Program

LASM syntax:

```
cale event number, lower threshold, upper threshold, hysteresis
```

Explanation:

The command sets the thresholds and hysteresis for the specified event number. The settings are calculated from the sensor measurements taken, and the sensor mode.

In order to secure "correct" parameters, several measurements are taken. From these measurements the average EventValue is calculated. No of measurements is as follows:

Physical sensors : 8 measurements
Virtual sensors : 1 measurement

When using the '**`cale`**' command, the program must be halted for some time, in order to allow the firmware to do the calibration.

To see how the actual calculations are done, please refer to the chapter Events, page 100.

The command is **not** useful for calibrating an event that monitors a switch sensor.

SetVar (0x14)

Command structure:

0x14	Variable number	Source	Value (LO)	Value (HI)

Reply:

0xE3

Availability:

Direct/Program

LASM syntax:

```
setv variable number, source, value
```

Explanation:

Sets the given variable with the given number.

SumVar (0x24)

Command structure:

0x24	Variable number	Source	Value (LO)	Value (HI)

Reply:

0xD3

Availability:

Direct/Program

LASM syntax:

```
sumv variable number, source, value
```

Explanation:

Adds the given number to the given variable.

SubVar (0x34)

Command structure:

0x34	Variable number	Source	Value (LO)	Value (HI)

Reply:

0xC3

Availability:

Direct/Program

LASM syntax:

```
subv variable number, source, value
```

Explanation:

Subtracts the given number from the given variable.

DivVar (0x44)

Command structure:

0x44	Variable number	Source	Value (LO)	Value (HI)

Reply:

0xB3

Availability:

Direct/Program

LASM syntax:

```
divv variable number, source, value
```

Explanation:

Divides the given variable with the given number.

If the given number is zero, the command is ignored.

MulVar (0x54)

Command structure:

0x54	Variable number	Source	Value (LO)	Value (HI)

Reply:

0xA3

Availability:

Direct/Program

LASM syntax:

```
mulv variable number, source, value
```

Explanation:

Multiplies the given variable with the given number.

SgnVar (0x64)

Command structure:

0x64	Variable number	Source	Value (LO)	Value (HI)

Reply:

0x93

Availability:

Direct/Program

LASM syntax:

```
sgnv variable number, source, value
```

Explanation:

Sets the given variable with the sign of the given number (-1, 0 or +1).

AbsVar (0x74)

Command structure:

0x74	Variable number	Source	Value (LO)	Value (HI)

Reply:

0x83

Availability:

Direct/Program

LASM syntax:

```
absv variable number, source, value
```

Explanation:

Sets the given variable with the absolute (positive) value of the given number.

AndVar (0x84)

Command structure:

0x84	Variable number	Source	Value (LO)	Value (HI)

Reply:

0x73

Availability:

Direct/Program

LASM syntax:

```
andv variable number, source, value
```

Explanation:

Does a bit-wise AND on the given variable with the given number.

OrVar (0x94)

Command structure:

0x94	Variable number	Source	Value (LO)	Value (HI)

Reply:

0x63

Availability:

Direct/Program

LASM syntax:

```
orv variable number, source, value
```

Explanation:

Does a bit-wise OR on the given variable with the given number.

Upload (0xA4)

Command structure:

0xA4	Datalog start (LO)	Datalog start (HI)	Datalog size (LO)	Datalog size (HI)

Reply structure:

0x53	Data point type [start]	Data point value (LO) [start]	Data point value (HI) [start]
	…	…	…
	Data point type [start + size - 1]	Data point value (LO) [start + size - 1]	Data point value (HI) [start + size - 1]

Availability:

Direct

LASM syntax:

```
polld start, size
```

Explanation:

Extracts data points from the data log (both size and start are measured in data points – not bytes) and puts them in the reply to the controlling PC. The first entry (index 0) has a type of 0xFF and the value contains the number of (valid) data points in the rest of the buffer.

See DatalogNext for a detailed explanation of the data points in the log.

SEnterEventCheck (0xB4)

Command structure:

0xB4	Event source	Event value (LO)	Event value (HI)	Bit 0-6: Jump Distance Bit 7: Jump direction 0 == forwards 1 == backwards

Availability:

Program

LASM syntax:

```
mone event source, event value, label or jump distance
```

Explanation:

Tells the firmware to inform the task when (any of) the given event(s) occur.

When the event occurs, execution continues at the given address and the monitoring is suspended.

Afterwards the actual event(s) can be read from source 23 if the task needs to distinguish between several different causes.

SetSourceValue (0x05)

Command structure:

0x05	Bit 0-5: Dest. Source	Dest. value	Bit 0-5: Orig. source	Value (LO)	Value (HI)

Reply:

0xF2

Availability:

Direct/Program

LASM syntax:

```
set destination source, destination value, origin source, origin value
```

Explanation:

The command is a general-purpose command that can be used to set any writable source inside the RCX.

It was introduced to avoid a plethora of specialized commands for setting the various new sources for e.g. event monitoring.

UnlockPBrick (0x15)

Command structure:

0x15	1	3	5	7	11

Reply structure:

0xE2	ROM Major (HI)	ROM Major (LO)	ROM Minor (HI)	ROM Minor (LO)	RAM Major (HI)	RAM Major (LO)	RAM Minor (HI)	RAM Minor (LO)

Availability:

Direct

LASM syntax:

```
pollp 1,3,5,7,11
```

Explanation:

The command replies with RCX firmware version information.

BeginOfTask (0x25)

Command structure:

0x25	0	Task number	Subroutine call list	Task size (LO)	Task size (HI)

Reply structure:

0xD2	Status: 0 == OK 1 == Not enough memory 2 == Illegal task number

Availability:

Direct

LASM syntax:

```
; there is no direct LASM syntax, this command is generated during download
```

Explanation:

The command reply tells whether there is memory enough and whether the task number is legal, so that the download may continue.

A number of ContinueDL commands is expected following a successful reply.

BeginOfSub (0x35)

Command structure:

0x35	0	Subroutine number	0	Subroutine size (LO)	Subroutine size (HI)

Reply structure:

0xC2	Status: 0 == OK 1 == Not enough memory 2 == Illegal subroutine number

Availability:

Direct

LASM syntax:

```
; there is no direct LASM syntax, this command is generated during download
```

Explanation:

The command reply tells whether there is memory enough and the subroutine number is legal, so the download may continue.

A number of ContinueDL commands is expected following a successful reply.

ContinueFirmwareDownLoad, ContinueDL (0x45)

Command structure:

0x45	Block count (LO)	Block count (HI)	Byte count (LO)	Byte count (HI)
	Data byte[0]			
	…			
	Data byte[Byte count – 1]			
	Block checksum			

Reply structure:

0xB2	Status: 0 == OK 3 == Block check sum error 4 == Firmware check sum error 6 == Download not active

Availability:

Direct

LASM syntax:

```
; there is no direct LASM syntax, this command is generated during download
```

Explanation:

The command reply tells whether there is memory enough or the task number is legal, so the download may continue.

Blocks are numbered 1, 2, …, n, 0 so the firmware can tell when the controlling computer believes the download is finished.

The block check sum is the sum of all the data bytes modulo 256.

GoIntoBootMode (0x65)

Command structure:

0x65	1	3	5	7	11

Reply:

0x92

Availability:

Direct

LASM syntax:

```
reset 1,3,5,7,11
```

Explanation:

Sends the RCX firmware into boot mode where it can mainly respond to very few commands and await the downloading of new firmware.

BeginFirmwareDownLoad (0x75)

Command structure:

0x75	Start address (LO)	Start address (HI)	Check sum (LO)	Check sum (HI)	0

Reply structure:

0x82	Status 0 == OK

Availability:

Direct

LASM syntax:

```
; there is no direct LASM syntax, this command is generated during download
```

Explanation:

The command reply tells whether there is memory enough so the firmware download can continue.

The start address is always 0x8000.

The firmware check sum is the sum of the first 19456 bytes in the firmware program (= 19 * 1024 = 19K) modulo 65536. If the size of the firmware file is less than 19 K,

A number of ContinueDL commands is expected following a successful reply.

SCheckDo (0x85)

Command structure:

0x85	Bit 0-5: Source 1 Bit 6-7: Comparison 0 == Greater than 1 == Less than 2 == Equal to 3 == Different from	Bit 0-5: Source 2	Value 1 (LO)	Value 1 (HI)	Value 2	Jump distance

Availability:

Program

LASM syntax:

```
chk src1, val1, relop, src2, val2, forward label or jump distance
```

Explanation:

If the comparison fails (evaluates to FALSE) execution jumps to the given address.

This can facilitate quite efficient implementation of switch/case statements because a failed match can jump forward to the next check.

LCheckDo (0x95)

Command structure:

0x95	Bit 0-5: Source 1 Bit 6-7: Comparison 0 == Greater than 1 == Less than 2 == Equal to 3 == Different from	Bit 0-5: Source 2	Value 1 (LO)	Value 1 (HI)	Value 2	Jump distance (LO)	Jump distance (HI)

Availability:

Program

LASM syntax:

```
chkl src1, val1, relop, src2, val2, forward label or jump distance
```

Explanation:

If the comparison fails (evaluates to FALSE) the execution jumps to the given address.

This can facilitate quite efficient implementation of switch/case statements because a failed match can jump forward to the next check.

UnlockFirmware (0xA5)

Command structure:

0xA5	'L'	'E'	'G'	'O'	'®'

Reply:

0x52	"Just a bit off the block!"

Availability:

Direct

LASM syntax:

```
boot 0x4C, 0x45, 0x47, 0x4F, 0xAE
```

Explanation:

If correct firmware has been downloaded, the firmware is then started.

LEnterEventCheck (0xB5)

Command structure:

0xB5	Event source	Event value (LO)	Event value (HI)	Bit 0-6: Jump Distance (LO) Bit 7: Jump direction 0 == forwards 1 == backwards	Jump Distance (HI)

Availability:

Program

LASM syntax:

```
monel event source, event value, label or jump distance
```

Explanation:

Tells the firmware to inform the task when (any of) the given event(s) occur.

When the event occurs, execution continues at the given address and the monitoring is suspended.

Afterwards the actual event(s) can be read from source 23 if the task needs to distinguish between several different causes.

ViewSourceValue (0xE5)

Command structure:

0xE5	0	Display precision (0-3) (decimal point position)	Display source	Display value (LO)	Display value (HI)

Reply:

0x12

Availability:

Direct/Program

LASM syntax:

```
disp precision, display source, display value
```

Explanation:

The command sets the user option for the LCD display. The LCD update process will then track the given data source and display it in the LCD display with the given decimal point position.

This command is very useful when debugging a program or if the user wants to display necessary application information that could otherwise only be communicated with sounds or by continuously polling the running program from a controlling computer.

Local variables cannot be shown in the display.

Motors

Motor control implements the following scheme:

Action	Result
Turning the RCX2 on	All motors are FORWARD, FLOAT and MAX power.
Turning the RCX2 off	All motors are FLOAT.
Start the selected program by pressing RUN == ”start 0”	All motors are FORWARD, BRAKE and MAX power.
Stopping the selected program by pressing RUN == ”stop” (stop all tasks)	All motors are BRAKE.
Changing the selected program by pressing PRGM == ”prgm N”	All motors are BRAKE.
Pressing motor control buttons on the LEGO remote control unit	Affected motors are FORWARD, ON and MAX power.
Releasing motor control buttons on the LEGO remote control unit	All motors are FORWARD, BRAKE and MAX power.
Pressing program select buttons on the LEGO remote control unit == ”prgm N NEWLINE start 0”	All motors are BRAKE.
Disrupting program flow by a higher priority task accessing shared resources	Affected motors are BRAKE.

A motor can be in six different states:

Event State	ON	OFF	FLOAT	FWD	RWD	ALT
ON_FWD	-	OFF_FWD	FLT_FWD	-	ON_RWD	ON_RWD
ON_RWD	-	OFF_RWD	FLT_RWD	ON_FWD	-	ON_FWD
FLT_FWD	ON_FWD	OFF_FWD	-	-	FLT_RWD	FLT_RWD
FLT_RWD	ON_RWD	OFF_RWD	-	FLT_FWD	-	FLT_FWD
OFF_FWD	ON_FWD	-	FLT_FWD	-	OFF_RWD	OFF_RWD
OFF_RWD	ON_RWD	-	FLT_RWD	OFF_FWD	-	OFF_FWD

The reason for explicitly stating the direction for a motor is that a 60 ms delay after each direction change has been introduced to reduce tear and wear on the motors if they are heavily loaded.

Events

The firmware will detect and report up to 16 independent events for physical and virtual sensors available to the programming system. The reporting usually takes the form of a signal/interrupt that changes the program flow for tasks that has asked to be alerted when an event occurs.

The 'sensors' for which events can be generated are:

- Physical sensors (Source 9, Range 0-2, Logical number 0-2)
- System timers (Source 1, Range 0-3, Logical number 3-6)
- IR Message (Source 15, Range 0, Logical number 7)
- Counters (Source 0/21, Range 0-2, Logical number 8-10)

The full range of events is:

- Pressed event (Event type 0)
- Released event (Event type 1)
- Period event (Event type 2)
- Transition event (Event type 3)
- Change-rate exceeded event (Event type 7)
- Enter low range (Event type 8)
- Enter normal range (Event type 9)
- Enter high range (Event type 10)
- Click (Event type 11)
- Double click (Event type 12)
- Mail box (Event type 14)

For each event, a number of attributes must be set from which the events are generated by comparison and change history. These attributes are:

- Upper threshold (Source 28)
- Lower threshold (Source 29) – these thresholds define the Low, Normal and High ranges above
- Hysteresis (Source 30) – defines how much trespassing is required before moving in/out of range
- Blink time (Source 31) – defines how slow/quick range changes must be to count as a Click event

In return, the firmware generates some event related information, which can be read (but not written) by the running program and polled from the controlling PC. The data sources are:

- Task events (Source 23) – bit mask that shows which events actually alerted the task
- Event state (Source 25) – low, normal or high
- Click counter (Source 27) – the number of Click events detected by the firmware for the event sensor

The data sources are all indexed by the event bit number and are specific to the event, not the sensor since a sensor can have multiple ranges defined by different events. This means that even though several events monitor the same sensor, each event must have set it thresholds and hysteresis individually.

Valid combinations

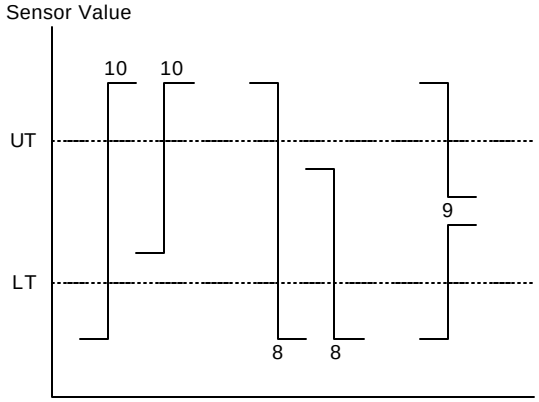
An event is a given sensor/event type combination. Not all types of events are available for all sensors. The possible combinations are indicated with '+' in the table below:

Sensor / Mode / Event type	Input 1, 2, 3 Absolute	Input 1, 2, 3 Dynamic	Timer 0-3 *	PBMessage *	Counter 0-2 *
Pressed	+				
Released	+				
Period	+	+			
Transition	+	+			
Change rate		+			
Enter low	+	+	+	+	+
Enter normal	+	+	+	+	+
Enter high	+	+	+	+	+
Click	+	+	+	+	+
Double click	+	+	+	+	+
Mail box				+	

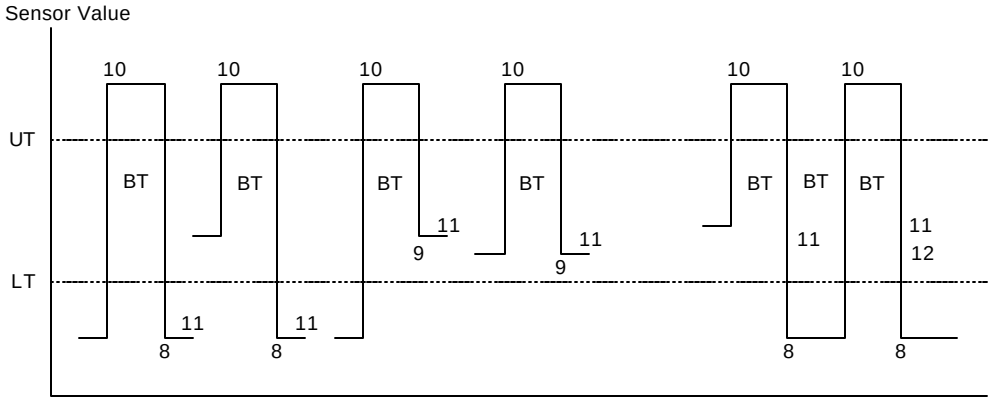
Event generation

The numbers printed on the following figures refer to the event type number in the table above.

- Enter High, Normal and Low events.

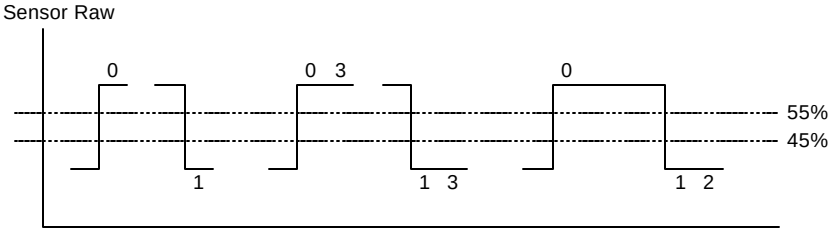


- Click and Double click events can/will lead to generation of several other events. A Double click is two Clicks with the correct timing between.



Where: Min. Duration < BT < Duration time + Min. Duration

- Pressed and Release events can will also generate other events, depending on the sensor mode.

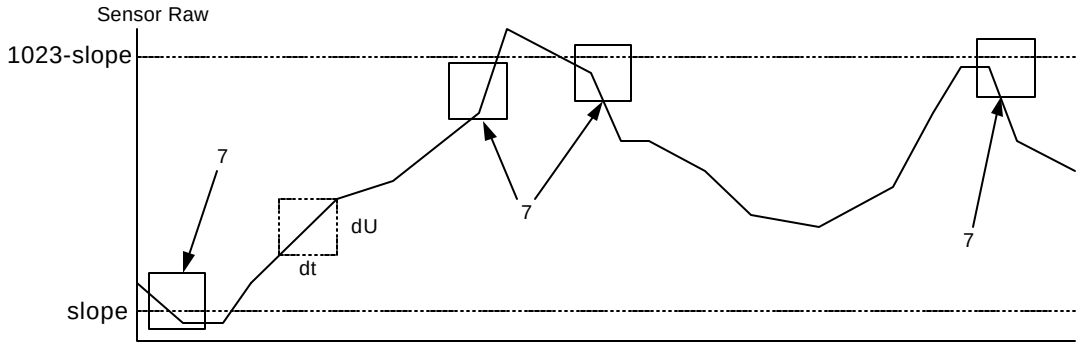


- Change-rate exceeded event. If using dynamic measurement the boolean value will hold information on the dynamic behavior of the sensor signal.

```
If  SensorRaw > Max - slope then FALSE
If  SensorRaw < slope then TRUE
Else
If dU/dt > slope then FALSE
If dU/dt < slope then TRUE
```

When the Boolean value changes, a Change-rate exceeded event is generated. The Boolean sensor value (source 13) will tell whether it was a rising (or positive) change rate or whether it was a falling (or negative) change rate as follows:

TRUE (1, one)	=> Negative/falling change rate
FALSE (0, zero)	=> Positive/rising change rate

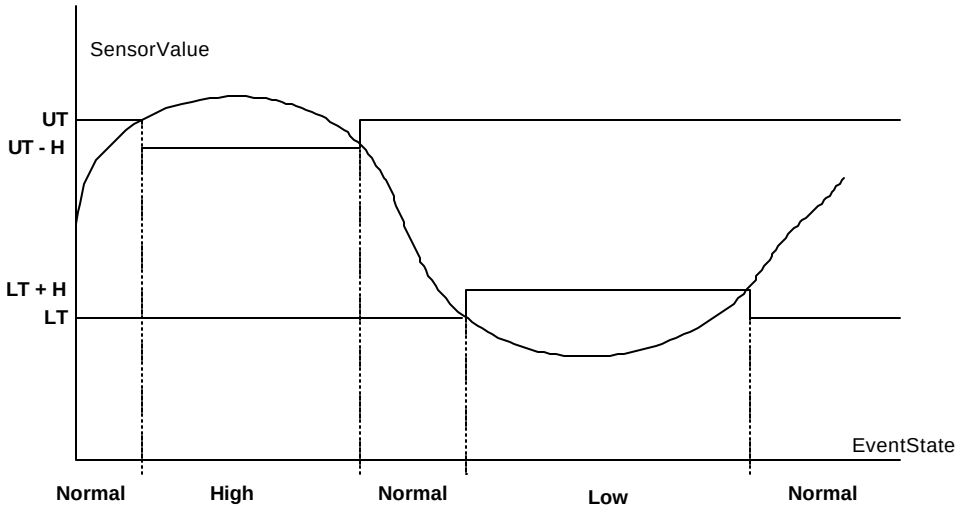


`dt` is fixed 3 ms, and `dU` can be chosen between 1 and 31 (Source 11).

- Mail box event is generated each time the RCX receives a message.

Thresholds and hysteresis

The thresholds are used together with the hysteresis to determine when the event changes state:



The hysteresis is used to limit spurious state transitions when the sensor value is close to a threshold.

Automatic calibration

To make sensor/event setup easier an automatic event calibration command is added that calculate the thresholds and hysteresis based on average sensor readings.

The calculations are dependent on the sensor mode. As the sensor mode can only be set for the physical sensors, these are the only exceptions. Below the calculations are listed for each (sensor) mode.

Temperature mode

$$UT = AvgValue + \frac{(AvgValue + 2730) * UTFct}{k1} + H$$

$$LT = AvgValue - \frac{(AvgValue - 2730) * LTFct}{k1} - H$$

$$H = \frac{AvgRaw * HFct}{k2}$$

$k1 = 4096, k2 = 1024$

Reflection mode

$$UT = AvgValue + \frac{(AvgValue * UTFct)}{k1} + H$$

$$LT = AvgValue - \frac{(AvgValue * LTFct)}{k1} - H$$

$$H = \frac{AvgRaw * HFct}{k2}$$

$k1 = 256, k2 = 2048$

Raw mode

$$UT = AvgValue + \frac{(AvgValue * UTFct)}{k1}$$

$$LT = AvgValue - \frac{(AvgValue * LTFct)}{k1}$$

$$H = \frac{AvgValue * Hfct}{k2}$$

$k1 = 256, k2 = 256$

All other modes

For Transition, Period, Angle, Boolean and all virtual sensors the calculations are:

$$UT = AvgValue + \frac{(AvgValue * UTFct)}{k1}$$

$$LT = AvgValue - \frac{(AvgValue * LTFct)}{k1}$$

$$H = 0$$

$$k1 = 256$$

Please note that it can be difficult to predict the outcome of a calibration. This is because the thresholds depend on the sensor values and the hysteresis, which again depend on the raw sensor value.

The values for k1 and k2 have been chosen to give suitable values for UT,LT and H.

Additionally commands for starting and stopping event monitoring will be added, based on the existing Scout commands.

An event setup command will be added that links a sensor to an event type and assigns an bit position for the event (out of a max total of 16). The assignment will be the responsibility of the user program.

Shared resources

In order to provide consistent dependable behavior a number of shared system resources, whose use should be controlled and monitored, have been identified:

- Motors
- Sound

The following enumeration for resources will be used

- 0x01 == Motor A,
- 0x02 == Motor B,
- 0x04 == Sound,
- 0x08 == Motor C (VLL output on the Scout)

The misalignment of the motors comes from the Scout heritage.

Resources are allocated by a changeable task priority.

There will be at least 8 priority levels with level 0 being the most important.

Tasks of equal priority can pre-empt each other. Although non-standard (and possibly the cause of a race-condition or mutual busy-waiting), it allows the user to assign equal priority to more tasks operating on the same resources and not cause them to wait if such behavior is easier to understand.

Access control & Event Monitoring

To guard shared resources, an access control mechanism has been added to protect tasks from accidentally taking control over other tasks resources. If all tasks follow the mechanism, deterministic priority-based behavior will be the result.

When event monitoring is used in combination with access control across several tasks, a complicated regime must be implemented as described in the following state machine:

EVENT STATE	EV	AC	EXIT EV	EXIT AC	ENTER EV (MON, LBL)	ENTER AC (RES, LBL)	START , STOP, DELETE TASK
NONE					IF MON<>0 EV=LBL **GO** EV ELSE **SEND** EXIT EV	IF **OK** AC=LBL RC=RES **GO** AC ELSE JUMP LBL	
EV	JUMP EV **GO** NONE		**GO** NONE		IF MON<>0 EV=LBL **GO** EV **(NEW)** ELSE **SEND** EXIT EV	IF **OK** AC=LBL RC=RES **GO** EV->AC ELSE JUMP LBL	**GO** NONE
AC		RELEASE RC JUMP AC **GO** NONE		**GO** NONE	IF MON<>0 EV=LBL **GO** AC->EV ELSE **SEND** EXIT EV	AC=LBL IF **OK** RC=RES **GO** AC **(NEW)** ELSE **SEND** AC	RELEASE RC **GO** NONE
EV->AC	RELEASE JUMP EV **GO** NONE	RELEASE RC JUMP AC **GO** EV	RELEASE RC **GO** NONE	**GO** EV	IF MON<>0 EV=LBL **GO** AC->EV **(ALERT)** ELSE **SEND** EXIT EV	AC=LBL IF **OK** RC=RES **GO** EV->AC **(NEW)** ELSE **SEND** AC	RELEASE RC **GO** NONE
AC->EV	JUMP EV **GO** AC	RELEASE RC JUMP AC **GO** NONE	 **GO** AC	**GO** NONE	IF MON<>0 EV=LBL **GO** AC->EV **(NEW)** ELSE **SEND** EXIT EV	AC=LBL IF **OK** RC=RES **GO** EV->AC **(ALERT)** ELSE **SEND** AC	RELEASE RC **GO** NONE

The events EV and AC are generated by the firmware and signal, respectively, that an event happened that the task was waiting for and that another task of equal or higher priority requested one or more of the resources currently held by the task.

Two state machine actions are used: GO and SEND. GO means that the state machine transitions to the indicated target state. SEND means that the indicated event is sent to the state machine (as an internal) event) – this typically constitutes commonality in transition behavior.

Returning from a subroutine that has started event monitoring or entered an access control region will result in either an “EXIT EV” or “AC” event to be sent to the state machine. The reason for this is that the flow of control might then later revert back **inside** the subroutine but **without** a return address.

Three state machine variables are used: EV, AC and RC. EV and AC are exit labels for event monitoring and access control respectively. RC denotes the resources currently held by a task.

The transitions marked **NEW** means that the old exit addresses and event/resource masks are overwritten and the ones marked **ALERT** signify that a nesting order was reversed. Both can potentially change the logic of the program, but whereas **NEW** transitions can be used to implement scoped nesting states, **ALERT** transitions can seriously change the result of later occurring requests and should be avoided if at all possible.

The **OK** condition, for access control, means that no other task (of a higher priority) held the requested resources. As a consequence of computing **OK**, other tasks (of equal or lesser priority) may receive an **AC** event.