Controlling

LEGO® Programmable Bricks

Technical Reference

Foreword

At LEGO, we believe that imagination is important to the world. For decades, the LEGO construction materials have been a means for people of all ages to express creativity and make discoveries of their own. The addition of LEGO programmable bricks brings a whole new dimension to construction.

The LEGO programmable bricks are microcomputers, which makes it possible to add functions or behaviour to physical creations made by LEGO pieces. The functions or behaviour are controlled by means of programming.

LEGO has launched two types of programmable bricks: the RCX(tm) of LEGO® MINDSTORMS(tm) Robotics Invention System(tm) and CyberMaster(tm) of LEGO Technic® CyberMaster(tm). The programming software codes of these two products have deliberately been designed to be easy to use -yet versatile and powerful in function. This has been important to enable kids to use the new technology for creation of their own personally meaningful inventions.

This technical reference guide is published to allow more creative freedom in the programming for more experienced users. The technical reference guide documents how the programmable bricks can be programmed by means of Visual Basic. We hope that the release of this document will inspire even more people to develop imaginative applications of the programmable bricks.

We kindly ask you to read the License Agreement and Warranty Disclaimer below before using this document.

We wish you good luck with development of creative applications.

LEGO - just imagine...

SOFTWARE DEVELOPER KIT LICENSE AGREEMENT

AND WARRANTY DISCLAIMER

License for the Software included in the LEGO MINDSTORMS Software Developer Kit (hereinafter referred to as the Software) from the LEGO Group.

IMPORTANT -- READ CAREFULLY: By using the information contained in this document you agree to be and are hereby bound by the terms of this License Agreement. If you do not agree to the terms of this Agreement, do not use the information contained in this document.

I. GRANT OF LICENSE:

The LEGO Group and its suppliers and licensors (hereinafter referred to as LEGO) hereby grant you a non-exclusive, non-commercial license to use the Software subject to the following terms:

You may:
- (i) use the Software only to develop applications for the LEGO MINDSTORMS RCX and the LEGO TECHNIC CYBERMASTER;
- (ii) the applications developed by means of the Software or parts hereof shall only be used for purposes that neither directly nor indirectly have any commercial implications;

You may not:

- (i) permit other individuals to use the Software except under the terms listed above;
- (ii) modify, translate, reverse engineer, decompile, disassemble (except to the extent that this restriction is expressly prohibited by law) or create derivative works based upon the Software;
- (iii) resell, rent, lease, transfer, or otherwise transfer rights to the Software; or
- (v) remove any proprietary notices or labels on the Software.

II. ENHANCEMENTS OR UP-DATES:

This license does not grant you any right to any enhancement or up-date.

III. TITLE:

Title, ownership, rights, and intellectual property rights in and to the Software shall remain with the LEGO Group. The Software is protected by national copyright laws and international copyright treaties. The communication protocol is protected by a pending patent application.

Title, ownership rights and intellectual property rights in and to the content accessed through the Software including any content contained in the Software media demonstration files is the property of the applicable content owner and may be protected by applicable copyright or other law. This license gives you no rights to such content.

LEGO, the LEGO logo, the LEGO Brick and LEGO MINDSTORMS are some of the trademarks belonging exclusively to the LEGO Group.

If you want to learn more about how to use trademarks and other proprietary rights belonging to the LEGO Group please visit our web site: http://www.lego.com.

“Visual Basic” is the trademark of Microsoft Corporation. “Delphi” and “C++ Builder” are the trademarks of Borland Corporation. All other trademarks are the property of their respective owners.

IV. DISCLAIMER OF WARRANTY:

THE SOFTWARE IS PROVIDED FOR FREE WITHOUT ANY KIND OF MAINTAINANCE OR SUPPORT.

THE SOFTWARE IS PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE LEGO GROUP FURTHER DISCLAIMS ALL WARRANTIES, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT. THE ENTIRE RISK ARISING OUT OF THE USE OR PERFORMANCE OF THE SOFTWARE OR APPLICATIONS DEVELOPED BY MEANS OF THE SOFTWARE REMAINS WITH YOU. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL THE LEGO GROUP OR ITS SUPPLIERS BE LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL, DIRECT, INDIRECT, SPECIAL, PUNITIVE, OR OTHER DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR OTHER PECUNIARY LOSS) ARISING OUT OF THIS AGREEMENT OR THE USE OF OR INABILITY TO USE THE PRODUCT, EVEN IF THE LEGO GROUP HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES/JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

V. TERMINATION:

This license shall terminate automatically if you fail to comply with the limitations described in this Agreement. No notice shall be required from the LEGO Group to effectuate such termination. On termination you must destroy all copies of the Software and applications developed by means of the Software.

VI. GOVERNING LAW:

This License Agreement shall be governed by the laws of the jurisdiction, where you have permanent residency. However, if the product is bought in USA the License Agreement shall be governed by the laws of the State of Connecticut, without regard to conflicts of law provisions, and if the product in bought in USA you consent to the exclusive jurisdiction of the state and federal courts sitting in the State of Connecticut. This License Agreement will not be governed by the United Nations Convention of Contracts for the International Sale of Goods, the application of which is hereby expressly excluded.

VII. ENTIRE AGREEMENT:

This Agreement constitutes the complete and exclusive agreement between the LEGO Group and you with respect to the subject matter hereof and supersedes all prior oral or written understandings, communications or agreements not specifically incorporated herein. This Agreement may not be modified except in writing duly signed by an authorised representative of the LEGO Group and you.

Introduction

This Technical Reference document tells you how to use SPIRIT.OCX directly to write LEGO MINDSTORMS RCX or LEGO Technic CyberMaster programs, providing more detailed control over the LEGO Programmable Bricks (abbreviated PBrick).

All examples in this document are written as Microsoft Visual Basic programs (version 5.0 – abbreviated VB) but other environments such as Borland Delphi/C++ Builder have been used successfully.

For program specific issues, the reader is referred to the respective manuals or online help systems. This document is not intended as a general introduction to programming in VB or other systems. The reader is assumed to be familiar with the different programming environments.

Pre-requisites

The SPIRIT.OCX ActiveX control must have been installed on the PC previously. This happens automatically when installing the LEGO MindStorms Robotics Invention System CD-ROM (1.0 or later – abbreviated RIS) or the LEGO Technic CyberMaster CD-ROM, so no further details will be given here.

Installing SPIRIT.OCX in VB

In the Components tab, tick “LEGO PBrickControl, OLE Control module” and drag an instance onto your main form. It appears as a LEGO logo and it looks really nice if you make it rectangular (495 x 495 VB units works quite well and is not too big). If it does not appear there, use the Add components feature and use the browser to find it.

Property settings

By selecting the object, you can set a number of useful properties for SPIRIT.OCX – in particular you can give it the name “PBrickCtrl”, which is useful if you want to use the examples in this document verbatim.

We have tried to make all names used in the examples be like the ones automatically generated by VB.

Name property

Allows you to change the name by which to refer to the control in the programs. We have used the name ‘PBrickCtrl’ throughout this document, but you may use any other name that you like, such as ‘PB’ if you don’t like typing.

PBrick property

Allows you to specify what kind of LEGO programmable brick you’re writing programs for. This property is used when checking the arguments of the methods in SPIRIT.OCX.

You can choose between “0 – Spirit” (for CyberMaster) and “1 – RCX”.

LinkType property

Allows you to specify the transmission type used to be either “0 – InfraRed”, “1 – Cable” or “2 – Radio”. This property tells SPIRIT.OCX how to format the data sent to the LEGO programmable brick and how to check the transceiver tower connected to the serial port of the PC.

Currently RCX uses “0 – InfraRed” and CyberMaster uses “2 – Radio” exclusively, so make sure you change both properties (PBrick and LinkType), when you change one of them. Obviously you must make sure that the hardware matches your settings.

ComPortNo property

Allows you to set the serial communications port that the PC uses to talk to the transceiver tower. The valid range is heavily PC hardware dependent, but usually include COM1 to COM4.

Be very careful, when changing this and look out for conflicts with modems, serial mouse devices, PDA hot-sync bays and other serial devices. You can use the LEGO MindStorms RIS Troubleshooting utility if this causes you any problems.

Program structure

A program consists of a number of tasks and subroutines executing in parallel (in a multitasking environment) and exchanging information via a set of common variables.

The downloaded programs are executed by an interpreter that carries out the instructions from the tasks that have been started and are ready to execute (not waiting).

The tasks are visited in a round robin fashion, so individual (byte code) instructions (commands) are executed atomically but the interleaving of tasks happens on a command-by-command basis.

The number of programs, tasks and subroutines vary between the RCX and CyberMaster. For further information see the parameter table on pg. TOBELINKED for details.

Tasks

The main structuring mechanism in RCX/CyberMaster programs are tasks which execute concurrently.

Starting a program by pressing the Run button on the RCX (Right button on CyberMaster) starts Task 0, which must then start all other tasks as required, possibly after setting up and initialising the system (setting input sensor types and modes, and setting outputs/motors to a known state).

Subroutines

To save program space, one can delegate common code to subroutines that can be called from the tasks.

Subroutines have no parameter and are shared between tasks. Several tasks can safely call the same subroutine at the same time.

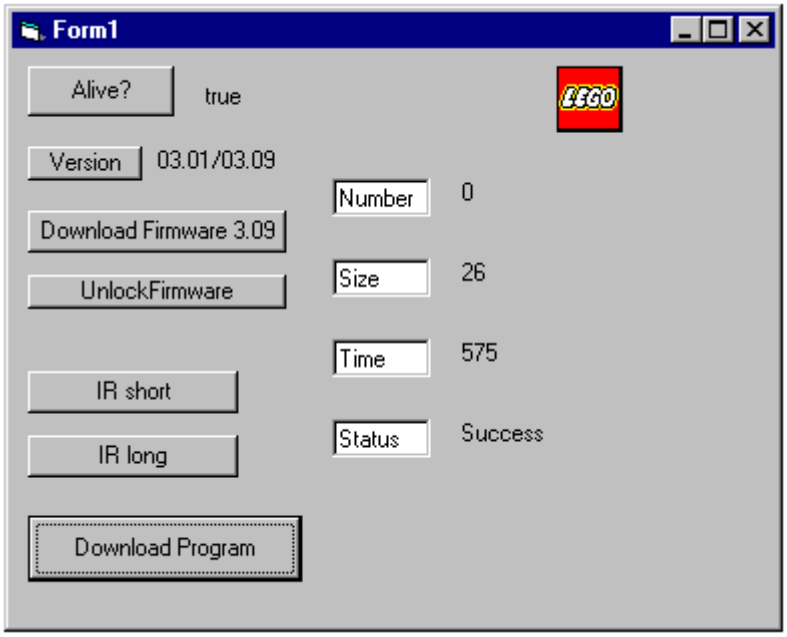
Variables

Variables in the RCX are more like general registers in a (RISC) microprocessor, in that they are addressed individually and they cannot be combined to form contiguous chunks of memory.

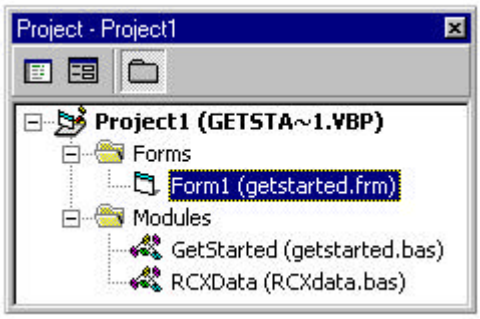
There are 32 variables (numbered 0-31) and they are shared between all tasks and subroutines. It is possible to implement a semaphore mechanism (using one global variable and exploiting the instruction set) to provide exclusive access to shared resources.

Example

The following example shows a small application that sets up the communication and presents a few buttons for querying the PBrick, setting a few properties and finally downloading a small program.The main form for the application looks like:



and the project includes a few header files (listed as appendices).



The program code for the form follows below:

Form1(code):

```
Private Sub AliveCheck_Click()
  If PBrickCtrl.PBAliveOrNot Then
    alive.Caption = "true"
  Else
    alive.Caption = "false"
  End If
End Sub

Private Sub DownPrgm_Click()
PBrickCtrl.SelectPrgm MotorControlProg
  PBrickCtrl.BeginOfTask MotorOnOffTask
    PBrickCtrl.Wait CON, 50
    PBrickCtrl.SetPower "motor0motor2", CON, kFullSpeed
    PBrickCtrl.SetFwd "motor0motor2"
    PBrickCtrl.On "motor0motor2"
    PBrickCtrl.Wait CON, 200
    PBrickCtrl.SetRwd "motor0motor2"
    PBrickCtrl.Wait CON, 200
    PBrickCtrl.Off "motor0motor2"
    PBrickCtrl.PlaySystemSound SWEEP_FAST_SOUND
  PBrickCtrl.EndOfTask
End Sub

Private Sub ShortIR_Click()
  PBrickCtrl.PBTxPower 0
End Sub

Private Sub Version_Click()
  FWver.Caption = PBrickCtrl.UnlockPBrick
End Sub

Private Sub LongIR_Click()
  PBrickCtrl.PBTxPower 1
End Sub

Private Sub UnlockFirmware_Click()
  PBrickCtrl.UnlockFirmware "Do you byte, when I knock?"
End Sub

Private Sub DownloadFirmware_Click()
  PBrickCtrl.DownloadFirmware "firm0309.lgo"
End Sub

Private Sub Form_Load()
  PBrickCtrl.InitComm
End Sub
```

Simple motor on wait off test program

Wait 0.5 sec.

Drive forward for 2 sec.

Wait 2 sec.
Change direction and drive 2 sec.
Wait 2 sec.

Play buildin sound

Initialise communication on start-up just in case. One could also use an extra command button for this.

```
Private Sub PBrickCtrl_DownloadDone(ByVal ErrorCode As Integer, ByVal DownloadNo As Integer)
  If ErrorCode = 0 Then
    D_stat.Caption = "Success"
    PBrickCtrl.PlaySystemSound SWEEP_DOWN_SOUND
  Else
    D_stat.Caption = "error: " + Chr(48 + ErrorCode)
  End If
End Sub

Private Sub PBrickCtrl_downloadStatus(ByVal timeInPBrickCtrl As Long,
                                      ByVal sizeInBytes As Long,
                                      ByVal taskNo As Integer)
  D_time.Caption = timeInPBrickCtrl
  D_size.Caption = sizeInBytes
  D_Nr.Caption = taskNo
End Sub
```

The form elements (labels, text boxes etc.) are not listed explicitly, but their names should be obvious from the code and intended functionality.

Good practice

Because all the SPIRIT.OCX methods use (constant) numbers to control the behaviour, we have found it useful to define most of these numbers (global constants) in a separate include file, and then have a separate include file for each project with specific settings.

The global constants make the programs more readable in general and the project specific constant definitions make the program understandable in terms of the problem it tries to solve (the robot it tries to control).

Table of Contents

OCX Overview

Communication control commands:

Firmware control commands:

Diagnostics commands:

PBrick system commands:

PBrick output control commands:

PBrick input control commands:

PBrick program control commands:

PBrick program execution commands:

PBrick flow control commands:

PBrick arithmetic/logical commands:

PBrick query commands:

PBrick data acquisition commands (RCX only):

ActiveX control commands:

ActiveX event dispatch interface:

Nomenklature:

☞	=	Immediate Command.
✎	=	Download-able Command.
R	=	For RCX only
C	=	For CyberMaster only
O	=	ActiveX (OCX) command, nothing transmitted to the PBrick
A	=	ActiveX asynchroneous events

ParameterTable #1/2

Command	Source (Number in cells below)																Motor-List	VarNo	RelOp	Time
	Var. (0)	Timer (1)	Const. (2)	Motor Status (3)	Ran-dom No. (4)	Tacho Counter C(5)	Tacho Speed C(6)	Motor Current C(7)	Prgm-No R(8)	Sensor-Value (9)	Sensor Type (10)	Sensor Mode (11)	Sensor Raw R(12)	Sensor Bool. R(13)	Watch R(14)	PB-Mes-sage R(15)	AGC C(16)			> 0 < 1 = 2 <> 3	
On(MotorList) **Off**(MotorList) **Float**(MotorList) **SetFwd**(MotorList) **SetRwd**(MotorList) **AlterDir**(MotorList) **SetPower**(MotorList, Source, Number)	0-31	•	0-7	•	0-7	•	•	•	•	•	•	•	•	•	•	•	•	0, 1, 2	•	•	•
C **ClearTachoCounter**(MotorList)	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	0, 1	•	•	•
SetEvent(Source, Number, Time) **ClearEvent**(Source, Number)	0	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	0-10,000 mS
Poll(Source, Number)	0 - 31	0 - 3	•	0, 1, 2	•	0, 1	0, 1	2	x	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2	0	0	0	•	•	•	•
SetVar(VarNo, Source, Number)	0-31	0-3	-32768 - 32767	0,1,2	1 - 32767	0,1	0,1	2	•	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2	0	0	•	•	0 - 31	•	•
SumVar(VarNo, Source, Number) **SubVar**(VarNo, Source, Number) **DivVar**(VarNo, Source, Number) **MulVar**(VarNo, Source, Number) **SgnVar**(VarNo, Source, Number) **AbsVar**(VarNo, Source, Number) **AndVar**(VarNo, Source, Number) **OrVar**(VarNo, Source, Number)	0 - 31	•	-32768 - 32767	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	0 - 31	•	•
Loop(Source, Number)	0 - 31	•	0 - 255	•	1 - 255	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
While(Source1, Number1, RelOp, Source2, Number2) **If**(Source1, Number1, RelOp, Source2, Number2)	0 - 31	0 - 3	-32768 - 32767	0, 1, 2	•	0, 1	0, 1	2	•	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2	0, 1, 2	0	0	•	0	•	0 - 3	•
Wait(Source, Number)	0 - 31	•	1 - 32767	•	1 - 32767	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
R **DatalogNext**(Source, Number)	0 - 31	0 - 3	•	•	•	•	•	•	•	0, 1, 2	•	•	•	•	0	•	•	•	•	•	•
R **SelectDisplay**(Source, Number)	0 - 31	•	0 - 6	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
R **SendPBMessage**(Source, Number)	0 - 31	•	0 - 255	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

R: For **RCX** only C: For **CyberMaster** only

ParameterTable #2/2

Command	Number	Time	Freq	Type (@) Only use for Poll of Sensor-Type (CyberMaster) 0: None 1: Switch 2: Temperature 3: Reflection 4: Angle 5: ID0 Switch (@) 6: ID1 Switch (@) 7: ID2 Switch (@)	Mode 0: Raw 1: Boolean 2: Trans. Counter 3: Period Counter 4: Percent 5: Celsius 6: Fahrenheit 7: Angle	Slope 0: Absolute 1-31: Dynamic	From	Size	Hours	Min.	Motorlist	Immidiate-Retries	Download-Retries
BeginOfTask(Number) **DeleteTask**(Number) **StartTask**(Number) **ResumeTask**(Number) **StopTask**(Number)	C: 0 - 3 R: 0 - 9	•	•	•	•	•	•	•	•	•	•	•	•
BeginOfSub(Number) **DeleteSub**(Number) **Gosub**(Number)	C: 0 – 3 R: 0 - 7	•	•	•	•	•	•	•	•	•	•	•	•
C **Drive**(Number0, Number1)	-7 -> 7	•	•	•	•	•	•	•	•	•	•	•	•
R **SetSensorType**(Number, Type) **ClearSensorValue**(Number) **SetSensorMode**(Number, Mode, Slope)	0, 1, 2	•	•	R: 0 – 4	C: 0 - 4 R: 0 - 7	0 – 31	•	•	•	•	•	•	•
PlayTone(Frequency, Time) **PlaySystemSound**(Number)	0 – 5	1-255 [10mS]	1-20,000	•	•	•	•	•	•	•	•	•	•
ClearTimer(Number)	0 – 3	•	•	•	•	•	•	•	•	•	•	•	•
PBPowerDownTime(Time)	•	1 – 255 [min] 0=forever	•	•	•	•	•	•	•	•	•	•	•
R **SelectPrgm**(Number)	0 - 4	•	•	•	•	•	•	•	•	•	•	•	•
R **SetWatch**(Hours, Minutes)	•	•	•	•	•	•	•	•	0 - 23	0 - 59	•	•	•
R **SetDatalog**(Size)	•	•	•	•	•	•	•	0 ~ delete log area 1 - available Memory	•	•	•	•	•
R **UploadDatalog**(From, Size)	•	•	•	•	•	•	0 – available Memory	1 - 50	•	•	•	•	•
R **PBTxPower**(Number)	0 - 1	•	•	•	•	•	•	•	•	•	•	•	•
C **OnWait**(MotorList, Number, Time)	-7 -> 7	0-255 [100mS !!]	•	•	•	•	•	•	•	•	0, 1, 2	•	•
C **OnWaitDifferent**(MotorList, Number0, Number1, Number2, Time)	-7 -> 7	0-255 [100mS !!]	•	•	•	•	•	•	•	•	0, 1, 2	•	•
SetRetransmitRetries(immidiateRetries, downloadRetries)							•					0-32767	1-32767

R: For RCX only. C: For **CyberMaster** only

☑ CyberMaster Command
☑ RCX Command

InitComm( )

☐ Downloadable Command
☑ Immediate Command

InitComm initialises the PC-Serial communication port.

The communication port (COM1, COM2, COM3 or COM4) can be selected via the property ComPortNo.

The type of transmitter link (CABLE, IR or RADIO) can be selected via the property LinkType.

Type of PBrick (CyberMaster or RCX) can be selected via the property PBrick.

Nothing is sent from the PC to the PBrick – it only sets up Spirit OCX.

This command should be used as the very first one (i.e. it initialises all communication features in Windows).

Part:	Description:
Return value:	If the initialisation succeeds, the return value is set to TRUE. If the initialisation fails, the return value is set to FALSE.

Example 1: **Values set via a property editor.**

```
If PBrickCtrl.InitComm Then
  Label1.Caption = “Comm-init OK”
Else
  Label1.Caption = “Init Comm FAILED”
EndIf
```

If the property ComPortNo is set to 1 this initialises the PBrick communication to COMM-port 1 (COM1).
LinkType and PBrick depends on the settings of the LinkType and PBrick Properties.

Example 2: **Values set by program code**

```
PBrickCtrl.LinkType = 2
PBrickCtrl.PBrick = 0
PBrickCtrl.ComPortNo = 3
Label1.Caption = PBrickCtrl.InitComm
```

Radio = 2 (IR = 0 and Cable = 1)
CyberMaster = 0 (RCX = 1)
Communication port COM3 selected.
Initialise the communication.

☑ CyberMaster Command
☑ RCX Command

CloseComm( )

☐ Downloadable Command
☑ Immediate Command

CloseComm closes the serialport so other applications can take over the port. E.g. low level debug tools.

Used when the user needs additional help for setting up his/her computer-system.

Example:

```
PBrickCtrl.InitComm

If PBrickCtrl.TowerAlive Then
  …
Else
  PBrickCtrl.CloseComm
End If
```

COM port initialised with the parameters set in the properties.

Tower is alive and H/W port works OK.

Ready for debug with low-level H/W debug code.

☑ CyberMaster Command
☑ RCX Command

GetShortTermRetransStatistics( )

☐ Downloadable Command
☑ Immediate Command

This command is used for checking the signal quality of the actual transmission. I.e. is the transmission disturbed by noise (CyberMaster) or external light (RCX)? Is the PBrick out of range? Is there an object between the sender and receiver (RCX)?

An increasing count of retransmission and/or signal corrections (CyberMaster) can signal the user application that there is some degrading of the signal quality. The user application can then decide what to do:

- Ask the user to move the PBrick closer to the Transceiver Tower. Manually (RCX/CyberMaster) or under joystick control (CyberMaster).

- If the PBrick is 'out of range' then a continuous retransmission will affect the communication between the PBrick and the user application. The retransmissions will lower the rate of information from the PBrick. Has the transmitted command(s) been received or not. The answer is delayed until the retransmission has finished. By lowering the setting of the retransmission rate, the application can get faster answers. To set the retransmission rate use the command **SetRetransmitRetries(ImmidiateRetries, DownloadRetries) -** see page 17 for further information. By default the retransmission rate is set to 5 and 10 (I.e. a total of 5 transmissions for immediate commands and a total of 10 for downloaded commands).

This command shows only information collected since last call. I.e. the info is automatically reset in the ActiveX control after each call. For long term transmission quality (from program start) see **GetLongTerm-RetransmitStatistics** on page 15.

Part:	Description:
Return value:	Variant. A two-dimensional OLE variant array. See the following page for a description of each element.

The returned information is positioned in the returned variant array as described on next page.

Element:	1st Dimension:	2nd Dimension:	Description:
1	Total count of transmitted commands since last call.	Total count of error corrected bytes since last call.	Total transmitted bytes normally 1 in noisy environments (I.e. function called after each command sent). Total error corrected bytes are showing the count of bytes in which one or more error correction(s) have been performed. Counted since last call.
2	Count of OK transmissions. Transmitted without any retransmission.	Count of answers Error corrected without any retransmission.	Commands without any retransmission. Succeeded in first transmission. Counted since last call. Count of corrected answers without any retransmission. Counted since last call.
3 - 12	Count of transmissions succeeded after 1- 10 retransmission.	Count of commands error corrected and succeeded after 1–10 retransmission.	Count of commands with 1-10 retransmission before success. Counted since last call. Count of commands succeeded when retransmitted 1-10 time and error corrected. Counted since last call.

Example:

```
Private Sub Command1_Click()
   Dim Stat As Variant
   Dim I As Long
   Dim val1 As Integer
   Dim val2 As Integer
   Stat =
PBrickCtrl.GetShortTermRetransStatistics()
   List1.Clear
   For I = LBound(Stat, 2)To UBound(Stat, 2)
     Val1 = Stat(0, I)
     Val2 = Stat(1, I)

   List1.AddItem Str(val) + " : " + Str(val2)
   Next I
End Sub
```

The value received as a safe array

Get size
Get ReTx element
Get ErrorCorrect Count
Display values
Repeat n times.

☑ CyberMaster Command
☑ RCX Command

GetLongTermRetransmitStatistics( )

☐ Downloadable Command
☑ Immediate Command

This command is used for checking the overall signal quality. The returned information is collected from program start.

A high count of retransmissions and/or signal corrections(CyberMaster) can be the result of an environment with a lot of electrical noise (CyberMaster) or room with much light (RCX). The user application can ask the user to:

- Move the PBrick away from the light (RCX). Place the transceiver tower away from the computer/monitor or other electrical equipment.

- If the PBrick is near 'out of range', the user application should ask the user, to move the PBrick closer to the Transceiver Tower.

The command shows information collected since program start. For short term transmission quality, see **GetShortTermRetransStatistics** on page 13.

Part:	**Description:**
Return value:	Variant. A two-dimensional OLE variant array. See the following page for a description of each element.

The returned information is positioned in the returned variant array as described on the following page.

Element:	1st Dimension:	2nd Dimension:	Description:
1	Total count of transmitted commands since program start.	Total count of error-corrected bytes since program start.	Total transmitted bytes since program start. Total error corrected bytes are showing the count of bytes in which one or more error correction(s) have been performed. Counted since program start.
2	Count of OK transmissions. Transmitted without any retransmission.	Count of answers error corrected without any retransmission.	Commands without any retransmission. Succeeded in first transmission. Counted since program start. Count of corrected answers without any retransmission. Counted since program start.
3 - 12	Count of transmissions succeeded after 1-10 retransmission.	Count of commands error corrected and succeeded after 1-10 retransmission.	Count of commands with 1-10 retransmission before success. Counted since program start. Count of commands succeeded when retransmitted 1-10 time and error corrected. Counted since program start.

Example:

```
 Private Sub Command1_Click()
    Dim Stat As Variant
    Dim I As Long
    Dim val1 As Integer
    Dim val2 As Integer
    Stat =
PBrickCtrl.GetLongTermRetransmitStatistics()
    List1.Clear
    For I = LBound(Stat, 2) To UBound(Stat, 2)
      Val1 = Stat(0, I)
      Val2 = Stat(1, I)

    List1.AddItem Str(val) + " : " + Str(val2)
    Next I
 End Sub
```

The value received as a safe array.

Get size
Get ReTx element
Get ErrorCorrect count
Display values
Repeat n times.

☑ CyberMaster Command
☑ RCX Command

SetRetransmitRetries(ImmidiateRetries, DownloadRetries)

☐ Downloadable Command
☑ Immediate Command

This command is used to fine-tune the rate of retransmission. The retransmission count can differ due to radio noise (CyberMaster), incoming light (RCX) or an out of range situation.

The command is very useful for getting an 'out of range' Pbrick back in contact. The return answer from the PBrick can be set in a "don't care" state. I.e. the application can continue to send motor commands to the PBrick and ignore the answers without the overhead of extensive retransmissions.

Part:	Description:
Return value:	If the function succeeds, the return value is set to TRUE. If the function fails, the return value is set to FALSE.
ImmidiateRetries:	An integer value for setting the retransmissions (total count of transmissions) for immediate commands. Range 1(0) - 32767. Default 5. The ImmidiateRetries can be set to zero (0) meaning the answer from the PBrick is totally ignored. This is useful when the PBrick gets "out of sight".
DownloadRetries:	An integer for setting the retransmissions while downloading program sequences and firmware (RCX). Range 1 to 32767. The default value is initially set to 10.

Example:

```
PBrickCtrl.SetRetransmitRetries( 1, 10 )
```

Need early warning if model cannot "hear" the commands. Normal retries when downloading.

☑ CyberMaster Command
☑ RCX Command

SetThreadPriority(threadNo, threadClass, ThreadPriority)

☐ Downloadable Command
☑ Immediate Command

This command is used to fine tune the performance of the user application/OCX. Both the process priority and the thread priority in the ActiveX component can be set/changed. Normally only the ActiveX threads should be manipulated.

The threadNo which is of the type THREADNAME is used to address the thread. ThreadClass which is of the type PROCESSPRIORITYCLASS addresses the priority class of the process. It should never be accessed with values other than DefaultClass = 0 (Nothing changes). ThreadPriority is a variable of the type THREADPRIORITY and it sets the priority of the thread addressed by threadNo.

The InitComm command should be used before this command.

Part:	**Description:**
Return value:	If the function succeeds, the return value is set to TRUE. If the function fails, the return value is set to FALSE.
threadNo	Is a variable of the type THREADNAME. In the ActiveX control the following threads are used: CommPortThread = 0 EventThread = 1 DownloadThread = 2 **Other values will be ignored.**
threadClass	Is a variable of the type PROCESSPRIORITYCLASS. This variable defines the different types of priority classes for the running process. Remember all the threads run in one single process. Changing this value for one process will alter the priority class for all the other threads. This is because this type of priority attaches to the process and not the thread. The default value is 3 (NORMAL_PRIORITY_CLASS). The following table shows the mapping between the internal priority class representation (values from 0 - 4) into the real win32 values:

Win32		Internal
DefaultClass	=	0
HIGH_PRIORITY_CLASS	=	1
IDLE_PRIORITY_CLASS	=	2
NORMAL_PRIORITY_CLASS	=	3
REALTIME_PRIORITY_CLASS	=	4

When the Set method is called with an 0 (DefaultClass) the ActiveX control will not try to alter the priority class for the process. This is **highly** recommended.

ThreadPriority Is a variable of the type THREADPRIORITY. Defines the thread priority for the threads. The following table shows the mapping between the internal thread priority representation (values from 0 - 6) into the real win32 values:

Win32		Internal
THREAD_PRIORITY_ABOVE_NORMAL	=	0
THREAD_PRIORITY_BELOW_NORMAL	=	1
THREAD_PRIORITY_HIGHEST	=	2
THREAD_PRIORITY_IDLE	=	3
THREAD_PRIORITY_LOWEST	=	4
THREAD_PRIORITY_NORMAL	=	5
THREAD_PRIORITY_TIME_CRITICAL	=	6

The default values for the three threads are:

Thread		Default value
CommPortThread	=	6
EventThread	=	5
DownloadThread	=	5

Example:

```
Dim pc As Integer
Dim pr As Integer
Dim name As Integer

name = Text1.Text
pc = 0

pr = Text2.Text
If PBrickCtrl.SetThreadPriority(name, pc, pr) Then
  Text3.Text = “Ok”
Else
  Text3.Text = “Error in SetThreadPriority”
End If
```

Don’t change the priority class for the process.

☑ CyberMaster Command
☑ RCX Command

GetThreadPriority(threadNo, threadClass, ThreadPriority)

☐ Downloadable Command
☑ Immediate Command

This command is used to get information about the current thread priority and performance of the user application/OCX. Both the process priority and the thread priorities in the ActiveX component can be read by this command

The threadNo which is of the type THREADNAME is used to address the thread. The priority class of the process is returned via the threadClass which is a pointer of the type PROCESSPRIORITYCLASS. The priority of the thread addressed by threadNo is returned via the ThreadPriority which is a pointer of the type THREADPRIORITY.

The InitComm command should be used before this command.

Part:	Description:

threadNo

Is a variable of the type THREADNAME. In the ActiveX control the following threads are used:

CommPortThread	=	0
EventThread	=	1
DownloadThread	=	2

If ThreadNo is out of range, the value of ThreadClass and ThreadPriority will be set to -1.

threadClass

Returns the value of the priority class of the process.

The default value is 3 (NORMAL_PRIORITY_CLASS). The following table shows the mapping between the internal priority class representation (values from 0 - 4) into the real win32 values:

Win32		Internal
DefaultClass	=	0
HIGH_PRIORITY_CLASS	=	1
IDLE_PRIORITY_CLASS	=	2
NORMAL_PRIORITY_CLASS	=	3
REALTIME_PRIORITY_CLASS	=	4

ThreadPriority Returns the value of the thread priority addressed by the ThreadNo.

The following table shows the mapping between the internal thread priority representation (values from 0 - 6) into the real win32 values:

Win32		Internal
THREAD_PRIORITY_ABOVE_NORMAL	=	0
THREAD_PRIORITY_BELOW_NORMAL	=	1
THREAD_PRIORITY_HIGHEST	=	2
THREAD_PRIORITY_IDLE	=	3
THREAD_PRIORITY_LOWEST	=	4
THREAD_PRIORITY_NORMAL	=	5
THREAD_PRIORITY_TIME_CRITICAL	=	6

The default values for the three threads are:

Thread		Default value
CommPortThread	=	6
EventThread	=	5
DownloadThread	=	5

Example:

```
Dim pc As Integer
Dim pr As Integer
Dim name As Integer

name = 0
PBrickCtrl.GetThreadPriority(name, pc, pr)
Label1.Caption = pc
Label2.Caption = pr
```

To get information about the CommPortThread the following VB code could be used (there is no check for failure).

CommPort

☐ CyberMaster Command
☑ RCX Command

DownloadFirmware(FileName)

☐ Downloadable Command
☑ Immediate Command

First it sets the PBrick in 'Boot Mode (acts like a PBrick without firmware). Next this command downloads the firmware version pointed to by 'FileName'.

This command will delete all downloaded tasks. The download may take a few minutes!

Part:	Description:
FileName:	Points to the file holding the firmware to be downloaded. The file name is a string holding both the PATH and the FILENAME following the normal operating system syntax.
Return value:	If the start-up of the download succeeds, the return value is TRUE, otherwise the return value is FALSE. The DownloadStatus event (see page 98) includes information about download size in bytes, the estimated download time, and the download type (here the type is "100", i.e. firmware download). The download is done in a separate thread in the ActiveX control, so the load on the control program should be minimal. Error(s) detected in the download thread will be sent to the user application as an event: The result of the firmware download will be sent in the DownloadDone event (See page 97). DownloadDone event message: **0:** Everything OK **1:** Download failed. Further information in the Asyncron-BrickError event (see page 99). Error code and description are both sent via this event.

Example:

`PBrickCtrl.DownloadFirmware "FIRM0309.LGO"`	Downloads the firmware to the PBrick, from the disk file FIRM0309.LGO

☑ CyberMaster Command
☑ RCX Command

IgnDLerrUntilGoodAnswer()

☐ Downloadable Command
☑ Immediate Command

The command sets an internal flag in the OCX. Used to flag a special transmission mode.

This command is used to recover from a transmission deadlock in a download. If the PBrick receives a command, considers it OK and replies with the normal Acknowledge it changes state and waits for the next block in the download.
But if the transceiver tower never receives this Acknowledge, the current transmission may come into a Deadlock (i.e. Transceiver Tower waits for the PBrick to reply and the PBrick waits for the next block in the download).

By issuing this command the download is set in a sort of "Error Tolerant" mode: The next program to download will start its download from the block that failed. This means that you have to download the same program.

Before starting the new download, the user should send a transmitted command (such as PBAliveOrNot) via the OCX. This is to synchronise the PBrick communication. It is very important to send a transmitted*) command before sending the new "BeginOfDownload" (Task or Sub).

*)Only real transmitted commands effects the togglebit. Host specific commands (e.g. InitComm) do not effect the togglebit.
PLEASE NOTE: Never send the DeleteTask command.

Example:

```
PBrickCtrl.BeginOfTask 0
.....many lines of code
EndOfTask



Label1 = PBrickCtrl.PBAliveOrNot
PBrickCtrl.IgnDLerrUntilGoodAnswer
PBrickCtrl.BeginOfTask 0
  ....
EndOfTask
```

Something disturbs the transmission between the PBrick and the Tower.
The AsyncronPBrickError returns "TOO_MANY_RESENDS" i.e. we can not receive an acknowledge.

New transmission started, the OCX continues until an acknowledge is received and the download then continues.

☑ CyberMaster Command
☑ RCX Command

UnlockPBrick( )

☐ Downloadable Command
☑ Immediate Command

This command is used for retrieving the ROM version. In the RCX PBrick the command also returns the version of the downloadable part of the firmware.

The ROM version is hardware specific for both the RCX- and the CYBERMASTER PBrick, while the firmware (RCX) is relating to the actual downloaded firmware in the RCX PBrick.

Part:	Description:
Return value:	A Basic string representing the version of the PBrick ROM and the downloadable firmware. An 11 character string is returned: **RR.rr/AA.aa** Where: **RR** represents the version of the ROM **rr** refers to the actual release of the ROM. **AA** represents the version of the downloadable part of the firmware. **aa** refers to the actual release of the downloadable part.

Example (RCX):

```
Label1.Caption = PBrickCtrl.UnlockPBrick
```

After the call the LABEL1 show:
03.02/03.09

ROM version 3, release 2.
The downloaded part of the System is version 3, release 9.

Example (CYBERMASTER):

```
Label1.Caption = PBrickCtrl.UnlockPBrick
```

After the call above LABEL1 shows:
01.01/00.00

ROM version 1, release 1

☑ CyberMaster Command
☑ RCX Command

UnlockFirmware(UnlockString)

☐ Downloadable Command
☑ Immediate Command

This command is used to open the command interpreter in the PBrick.

RCX: The UnlockFirmware command should be issued after each new download of the downloadable firmware (e.g. after battery change or update of software).

CyberMaster: The UnlockFirmware command should be sent after each power up. The command also sets the powerdown time to 10 min. implicitly.

Part:	**Description:**
UnlockString:	The 1st part of LEGO copyrighted unlock handshake string: "Do you byte, when I knock?"
Return value:	Users trying to use another UnlockString are forced to display the LEGO logo. A DirectX application will not show the logo, but the MessageBox pops up behind the active DirectX application and freezes the application. The user can use a correct UnlockString to prevent the MessageBox from popping up, but then he has to use the LEGO copyrighted text string.

Example 1 (Correctly supplied UnlockString:)

```
Label1.Caption = PBrickCtrl.UnlockFirmware(“Do you byte, when I knock?”)
```

This example will set the label Label1.Caption to “This is a LEGO Control OCX communicating with a LEGO PBrick!” if the UnlockFirmware succeeds else the LABEL1.Caption is set to: “Unlock failed”.

Example 2 (Invalid UnlockString supplied:)

```
Label1.Caption = PBrickCtrl.UnlockFirmware(“XYZ”)
```

This example will set the label LABEL1.Caption to “The LEGO Control OCX can not get a valid PBrick Unlock string!” and the MessageBox will pop up or block the application (DirectX).

☐ CyberMaster Command
☑ RCX Command

SelectDisplay(Source, Number)

☑ Downloadable Command
☑ Immediate Command

This command is used to perform the same operation as pressing the 'Select key' on top of the RCX PBrick.

Part:	Description:

Source Number:

Source can be either a VAR or a CONSTANT type.
If Source is a VAR then the range for Number is 0-31.
If Source is a CONSTANT the range for Number is 0-6.
If the selected VAR (0-31) contains a value <0 or >6 then the SelectDisplay-command is ignored.

Constant pointed to by Number or value in Var (0-31):	Display shows:
0	SoftwareWatch
1	Input 0 (labeled 1 on the box)
2	Input 1 (labeled 2 on the box)
3	Input 2 (labeled 3 on the box)
4	Output 0 (labeled A on the box)
5	Output 1 (labeled B on the box)
6	Output 2 (labeled C on the box)

Return value:

If the function succeeds, the return value is TRUE.
If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SelectDisplay 2, 5
```

The display on the PBrick will show the power setting of output "1". E.g. a displayed 4 equals power level 4.

☐ CyberMaster Command
☑ RCX Command

SetWatch(Hours, Min)

☑ Downloadable Command
☑ Immediate Command

Sets the PBrick Software Watch. The watch is a 24-hours type (i.e. 0 to 23:59).

Part:	Description:
Hours:	Sets the Hour part of the PBrick's internal software watch (0 - 23).
Min:	Sets the Minute part of the PBrick's internal software watch (0 - 59).
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SetWatch 13, 22
```

Sets the internal software watch to 13:22.

☑ CyberMaster Command
☑ RCX Command

MemMap( )

☐ Downloadable Command
☑ Immediate Command

Returns an variant (i.e. a safe array) of memory pointers.

Part:	Description:	
Return value:	A complete memory map in a variant (safe array).	
Array elements **(RCX):**	**ErrorCode**	Inverted MemMap command for <OK> and 0x00 flagging a <FAIL>
	Sub00, ...Sub07, **Sub10, ...Sub17,**	Sub00 = pointer to <Program0, Sub0> Sub17 = pointer to <Program1, Sub7>
	-	
	Sub40, ...Sub47,	Sub47 = pointer to <Program4, Sub7>
	Job00, ...Job09,	Job00 = pointer to <Program0, Job0>
	-	
	Job40, ...Job49,	Job49 = pointer to <Program4, Job9>
	Log,	Pointer to start of Datalog Area
	CurrentLog,	Most recent datalog
	MemUsed,	Total of mem used (incl. allocated datalog area)
	MemTop	Pointer to last byte in UserMemory
Array elements **(CyberMaster):**	**ErrorCode**	Inverted MemMap command for <OK> and 0x00 flagging a <FAIL>
	Sub00, ...Sub03,	Sub00 = pointer to <Program0, Sub0> Sub03 = pointer to <Program0, Sub3>
	Job00, ...Job03,	Job00 = pointer to <Program0, Job0> Job03 = pointer to <Program0, Job3>
	MemUsed,	Total of mem used (incl. allocated datalog area)
	MemTop	Pointer to last byte in UserMemory

All elements are pointers represented as 16-bit signed integers (due to the OLE/safe array interface.) The size of any element can be calculated as: (Ptr to next element) – (Ptr to this element).

ErrorCode Is a 16-bit flag showing OK or FAIL. Inverted MemMap command means an OK MemMap returned. A "0x00" flags an error.

SubXY Points at the start address of subroutine Y in program X.

JobXY Points at the start address of task Y in program X.

Log The element points at the start of the datalog area.

CurrentLog The element points to the last element currently logged.

MemUsed This element points to the last allocated byte in the user ram. Hence, it also points to the last byte in the datalog area.

MemTop Points to the last available byte in user ram. I.e. MemTop and User Memory Size are identical.

Example

MemMapListbox List of "pointers elements" StatusLabel - Shows result of operation (error code).

```
Private Sub Command1_Click()
  Dim Stat As variant
  Dim I As Integer
  Dim MemValue As Integer

  Stat = PBrickCtrl.MemMap
  MemValue = Stat(LBound(Stat))
  Label1.Caption = Str(MemValue)
  List1.Clear
  For I = LBound(Stat) + 1 To
          UBound(Stat)
    MemValue = Stat(I)
    List1.AddItem Str(MemValue)
  Next I
End Sub
```

Data returned in a safe array

First element = ErrorCode
Show status
Clean up the listbox.

Iterate over the elements

☑ CyberMaster Command
☑ RCX Command

PBPowerDownTime(Time)

☐ Downloadable Command
☑ Immediate Command

Sets the PBrick's Auto PowerdownTime. (The PBrick's default Powerdown time is 15 minutes).

Part:	Description:
Time:	0 - 255. The PowerdownTime is set in minutes [1 to 255]. If Time is set to "0" (zero) it means auto powerdown is disabled (i.e. the PBrick is on forever). See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.PowerDownTime 120
```

The PBrick's auto powerdown time is set to 2 hours.

☑ CyberMaster Command
☑ RCX Command

PBBattery( )

☐ Downloadable Command
☑ Immediate Command

PBrick Battery check. The function returns the voltage of the PBrick battery. The value is an average sampled over the last 30 seconds. I.e. a start-up of a motor will not affect the value.

Part:	Description:
Return value:	Voltage returned in a signed integer (short). If the function succeeds, the return value is the battery voltage in mV. Otherwise a zero (0) is returned. If the PBrick is not accessible (e.g. Turned off) an error will be issued. The user application should have an error handler, e.g. Visual Basic On Error Goto <lblMyErrorHandler>.

Example:

```
Label1.Caption = PBrickCtrl.PBBattery()
```

Sets label ”Label1” to the voltage level of the PBrick’s batteries [millivolts].

☑ CyberMaster Command
☑ RCX Command

PBAliveOrNot( )

☐ Downloadable Command
☑ Immediate Command

Use this command as a quick way to determine whether the PBrick is able to answer or not (The whole set-up, SW, Cable, transceiver tower, and PBrick is tested).

Part:	Description:
Return value:	Boolean. If the PBrick is alive and within range, the return value is set to TRUE. If no answer is received from the PBrick within the timeout period, the return value is set to FALSE.

Example:

```
If PBrickCtrl.PBAliveOrNot Then
  Label1.Caption = “OK communication”
Else
  Label1.Caption = “Unable to communicate with the PBrick”
EndIf
```

☑ CyberMaster Command
☑ RCX Command

PBTurnOff( )

☑ Downloadable Command
☑ Immediate Command

The PBrick stops all running jobs, and turns itself off.

Part:	Description:
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
If PBrickCtrl.PBTurnOff Then
  Label1.Caption = “PBrick Turned OFF”
Else
  Label1.Caption = “PBrick Status UnKnown”
EndIf
```

If the PBrick is turned off correctly the returned value is TRUE.

☐ CyberMaster Command
☑ RCX Command

PBTxPower(Number)

☑ Downloadable Command
☑ Immediate Command

Set the IR transmitter power of the RCX PBrick. If more than one RCX is used in the same room, the TxPower should be set to Low, to prevent the RCXs from interfering with each other. (The transmitter power of the transceiver tower has to be set manually with the switch on the front of the tower).

Part:	Description:
Number:	0: Short range mode 1: Long range mode.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
If (PBrickCtrl.PBTxPower 1) Then

  Label1.Caption = “PBrick setup for LONG range”
Else
  Label1.Caption = “PBrick Status UnKnown”

EndIf
```

The PBrick is forced into LONG range tx.

☑ CyberMaster Command
☑ RCX Command

TowerAndCableConnected()

☐ Downloadable Command
☑ Immediate Command

Used to detect a correctly connected cable and transceiver tower. The command uses the “Request To Send” (RTS) and the “Clear To Send” (CTS) signal to check for the hardware connection.

The command toggles the RTS line and the signal is shortcircuited to the CTS line by a jumper in the transceiver tower. The function detects the transition of the CTS line, so a cabling failure should not give a wrong signal (except a shortcircuited RTS/CTS).

Part:	**Description:**
Return value:	If the cabling is OK, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
If PBrickCtrl.TowerAndCableConnected Then
  Label1.Caption = “Hardware cabling OK”
Else
  Label1.Caption = “Hardware FAIL! Please control the cabling.”
End If
```

☐ CyberMaster Command
☑ RCX Command

TowerAlive()

☐ Downloadable Command
☑ Immediate Command

Used to check the status of the transceiver tower. Is the hardware and battery OK? Due to the fact that the infrared Receiver can “see” the transmitted signal, the input buffer of the serial channel will contain a copy of the transmitted bytes. This optical feedback can be used as an alive check of the Tower. The test is also a simple check of the battery in the TxTower.

This command always first checks the total transmission by sending an implicit PBAliveOrNot. If the PBAliveOrNot command returns FALSE, a special transmission string is sent to the transceiver tower. The crosstalk in the transceiver tower is used for this test. If this test fails the string is only sent twice, not with the RetransmitRetries setting.

IMPORTANT: Should always be used after a TowerAndCableConnected command. A short-circuited cable and/or crosstalk in the hardware could affect the result (i.e. give a TRUE without any tower connect).

Part:	**Description:**
Return value:	If Tower is alive (i.e. battery and hardware OK), the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
If PBrickCtrl.TowerAlive Then
  Label1.Caption = “Tower hardware and battery OK”
Else
  Label1.Caption = “Tower H/W-FAIL or Tower battery should be changed!”
End If
```

☑ CyberMaster Command
☑ RCX Command

SetEvent(Source, Number, Time)

☐ Downloadable Command
☑ Immediate Command

Sets and enables the autopolling feature. When enabled the OLE control automatically polls Variable 0 (zero) of the PBrick with the time interval set by the value of **Time**. If a change of Variable 0 (zero) is detected, the ActiveX control automatically sends an OLE event to the application.

Hence, the impact on the user application is minimal.

All internals in the PBrick can be scanned. The downloaded program should include a small routine which sends or refreshes the required data into Variable 0.

Part:	**Description:**
Source, Number:	Source and Number addresses what to autopoll for. See ParameterTable for ranges, page 9. (Currently only implemented for Variable 0)
Time:	Sets the time interval for the autopoll (in ms.).
Return value:	If the function succeeds (i.e. the set-up of the event in the ActiveX control), the return value is TRUE. If the function fails, the return value is FALSE.

Example:

`PBrickCtrl.SetEvent( Var, 0, 300 )`	Set up autopolling on variable 0, with a time interval of 300ms.

If any changes in variable occur, the OLE event VariableChange will be sent to the application, reflecting the new value for Variable 0. (See page 96)

☑ CyberMaster Command
☑ RCX Command

ClearEvent(Source, Number)

☐ Downloadable Command
☑ Immediate Command

Clears an event set up by SetEvent (see page 37).

Part:	Description:
Source, Number:	Source and Number point out what to disable autopoll for. See ParameterTable for ranges, page 9. (Currently only implemented for Variable 0).
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

☑ CyberMaster Command
☑ RCX Command

BeginOfTask(Number)

☐ Downloadable Command
☑ Immediate Command

Initialises a task download sequence. All commands following this, and until the EndOfTask or EndOfTaskNoDownload commands will be buffered in the ActiveX control. The actual compilation and download of the code to the PBrick will start when the EndOfTask or EndOfTaskNoDownload command is received.

When the download is finished, the ActiveX control will send the 'DownloadDone' event with information about the result of the download. See page 97 for more information about the DownloadDone event.

Part:	Description:
Return value:	If start-up of buffering commands is started OK, the return value is TRUE. If the function fails, the return value is FALSE.

Example This example downloads a program to task 1:

```
PBrickCtrl.BeginOfTask 1      Initialises download to task 1
  PBrickCtrl.On "01"          Starts motor 0&1
  PBrickCtrl.Wait 100         Stops program execution for 1 sec.
  PBrickCtrl.Off "01"         Stops motor 0&1
PBrickCtrl.EndOfTask          Ends the download sequence.
```

When the EndOfTask command is reached, the program buffered in the ActiveX control is compiled, syntax checked and transferred to the PBrick via the transceiver tower.

Estimated download time and compiled size of the task will be sent in the DownloadStatus event, see page 98.

☑ CyberMaster Command
☑ RCX Command

EndOfTask( )

☐ Downloadable Command
☑ Immediate Command

Part of BeginOfTask…EndOfTask sequence.

For further details see **BeginOfTask**.

Part:	**Description:**
Return value:	Short type. If the startup of this command, i.e. the compiler and download started OK, the return value = 1. Any error in the compile- and download thread will result in a return value of 0.

Further info returned in:

DownloadStatus:	The user will receive info about the task size (compiled), estimated download time and task number in this event.
DownloadDone:	When the download is finished the user will get this event. A non zero-value (i.e. 1) indicates that an error occurred. A value of zero (0) indicates that the transfer of the task was OK. If a 1 is received further information is given in the AsyncronBrickError event. To synchronise these two events in a single threaded application (e.g. Visual Basic) some user synchronisation should be implemented: See Appendix A on page 102 for further details.
AsyncronBrickError:	This event sends an ErrorNumber and an ErrorDescription

Example 1:

```
PBrickCtrl.BeginOfTask 1
...
EndOfTask
```

Initialises download to task 1

Ends the download sequence.

Example 2:

```
PBrickCtrl.BeginOfTAsk 4
  PBrickCtrl.On “01”
  PBrickCtrl.SetFwd “2”
EndOfTask
```

Initialises download to task 4

Ends the download sequence.

☑ CyberMaster Command
☑ RCX Command

EndOfTaskNoDownload( )

☐ Downloadable Command
☑ Immediate Command

This command is used to gain info about the compiled and downloadable task in advance of the real download.

If the amount of free RAM in the PBrick is small, the user can check for needed space by using this command and the MemMap command, see page 28 for information about MemMap.

Part:	Description:
Return value:	Short type. If the startup of this command, i.e. the compiler and download started OK, the returned value is 1. Any error in the compile thread will result in a return value of 0.

Further information returned in:

DownloadStatus:	The user will receive info about the task size (compiled), esitmated download time and task number in this event.

Example:

Example 1: (The download function)

```
PBrickCtrl.BeginOfTask 1
  ...
PBrickCtrl.EndOfTaskNoDownload
```

Storage for memory info
Initialises download to task 1

Ends the pseudo-download and get the block-size etc. in the DownLoadStatus event.

Example 2: (The download info returned in the DownloadStatus event)

```
PBrickCtrl_DownloadStatus  (ByVal DownloadTimeInMS As Long,
                            ByVal sizeInBytes As Long,
                            ByVal taskNo As Integer)

Label1.Caption = DownloadTimeInMS
Label2.Caption = sizeInBytes
Label3.Caption = taskNo
```

Expected download time.
Size of the compiled task in bytes.
Number of the task.

☑ CyberMaster Command
☑ RCX Command

BeginOfSub(Number)

☐ Downloadable Command
☑ Immediate Command

Initialises a subroutine download sequence. All commands following this, and until the EndOfSub or EndOfSubNoDownload command will be buffered in the ActiveX control. The real download to the PBrick is started when the EndOfSub and EndOfSubNoDownload commands is reached.

When the download is finished, the ActiveX control will send the 'DownloadDone' event with information about the result of the download. See page 97 for more information about the DownloadDone event.

Part:	Description:
Return value:	If the function succeeds , the return value is TRUE. If the function fails, the return value is FALSE.

Example: This example downloads a program to Subroutine 1.

```
PBrickCtrl.BeginOfSub 1
  PBrickCtrl.Loop 2, 8
    PBrickCtrl.SumVar 3, 0, 2
  PBrickCtrl.EndLoop
PBrickCtrl.EndOfSub
```

Initialises download to Subroutine 1.
Loops 8 times through the following block.
Adds two to Variable 3.
Ends the loop.
Ends the download sequence.

When EndOfSub command is reached, the program buffered in the ActiveX control will be compiled, syntax checked and transferred to the PBrick via the transceiver tower.

☑ CyberMaster Command
☑ RCX Command

EndOfSub( )

☐ Downloadable Command
☑ Immediate Command

Part of BeginOfSub…EndOfSub sequence.

For details see **BeginOfSub** on page 42.

Part:	Description:
Return value:	Short type. If the startup of this command, i.e. the compiler and download started OK, the return value = 1. Any error in the compile- and download thread will result in a return value of 0.

Further info returned in:

DownloadStatus:	The user will receive info about the subroutine size (compiled), estimated download time and task number in this event.
DownloadDone:	When the download is finished the user will get this event. A non-zero value (i.e. 1) indicates that an error occurred. A value of zero (0) indicates that the transfer of the task was OK. If a value of 1 is received further information is given in the AsyncronBrickError event. To synchronise these two events in a singlethreaded application (e.g. Visual Basic) some user synchronisation should be implemented. See Appendix A on page 102 for further details.
AsyncronBrickError:	This event sends an ErrorNumber and an ErrorDescription

Example 1:

`PBrickCtrl.BeginOfSub 1`	Initialises download to subroutine 1
`...`	
`PBrickCtrl.EndOfSub`	Ends the download sequence.

Example 2:

`PBrickCtrl.BeginOfSub 1`	Initialises download to subroutine 1
`  PBrickCtrl.On “1”`	
`  PBrickCtrl.AlterDir “2”`	
`  PBrickCtrl.PlaySystemSound 2`	
`PBrickCtrl.EndOfSub`	Ends the download sequence.

☑ CyberMaster Command
☑ RCX Command

EndOfSubNoDownload( )

☐ Downloadable Command
☑ Immediate Command

This command is used to gain info about the compiled and downloadable subroutine in advance of the real download.

If the amount of free RAM in the PBrick is small, the user can check for needed space by using this command and the MemMap command, see page 28 for information about MemMap.

Part:	Description:
Return value:	Short type. If the startup of this command, i.e. the compiler and download started OK, the returned value = 1. Any error in the compile- and download thread will result in a return value of 0.

Further info returned in:

DownloadStatus:	The user will receive info about the subroutine size (compiled), estimated download time and subroutine number in this event.

Example 1: (The download function)

```
PBrickCtrl.BeginOfSub 1
  ...
PBrickCtrl.EndOfSubNoDownload
```

Storage for memory info
Initialises download to subroutine 1

Ends the pseudo-download and get the block size etc. in the DownloadStatus event.

Example 2: (The download info returned in the DownloadStatus event)

```
PBrickCtrl_DownloadStatus (ByVal DownloadTimeInMS As Long,
                           ByVal sizeInBytes As Long,
                           ByVal taskNo As Integer)

Label1.Caption = DownloadTimeInMS
Label2.Caption = sizeInBytes
Label3.Caption = taskNo
```

Expected download time.
Size of the compiled subroutine in bytes.
Number of the subroutine.

☑ CyberMaster Command
☑ RCX Command

Poll(Source, Number)

☐ Downloadable Command
☑ Immediate Command

This command is used to retrieve information/status from the PBrick. E.g. Variables, Timers, Input information, Motor status etc.

Part: **Description:**

Source, Number: Source and Number is used to address what to 'Poll'.
See ranges in the ParameterTable on page 9.

Source:	Number:	Description:
0	0-31	Variable 0-31.
1	0-3	Timer 0-3.
2	-	-
3	0,1,2	Motor status. The information is packed: Bit 7: ON/OFF 1/0 Bit 6: Brake/Float 1/0 Bit 5: Output no. HiBit Bit 4: Output no. LoBit Bit 3: Direction CW/CCW 1/0 Bit 2: PowerLevel Most significant bit Bit 1: PowerLevel Bit 0: PowerLevel Least significant bit
4	-	-
5	0,1	CyberMaster. TachoCounts. The Tacho value is approx. 50 counts per revolution of the shaft.
6	0,1	CyberMaster. Tacho Speed. Normally a value between 0 and 90 [relative proportional value].
7	2	CyberMaster. MotorCurrent [milli amps, approx. value], only valid for the external motor 2.
8	-	RCX. Program No. I.e. Actual program selected.
9	0,1,2	SensorValue. Value measured at an input. Depends on the actual mode of operation.
10	0,1,2	SensorType. Tells what type of sensor the input is set-up for.
11	0,1,2	SensorMode. Tells what mode the input is set-up for.
12	0,1,2	RCX. SensorRaw i.e. the analogue value measured at the input.
13	0,1,2	RCX. SensorBoolean. Returns the Boolean state of the input.
14	0	RCX. Watch. Integer where MSB = hours and LSB = minutes.
15	0	RCX. Returns the PBMessage stored internally in the RCX.
16	-	CyberMaster. The AGC (automatic gain control) voltage from the CyberMaster Base Unit [in millivolt].

Return value: The return value is the data/status asked for (16 bit signed Integer).
If the PBrick is not accessible (e.g. Turned off) an error will be issued. The user application should have an error handler, e.g. Visual Basic On Error Goto <lblMyErrorHandler>.

Example:

`Label1.Caption = PBrickCtrl.Poll 0, 7`	Label1 will be set equal to variable 7 of the PBrick.
`Label2.Caption = PBrickCtrl.Poll 7, 2`	Label2 will be set equal to the MotorCurrent of the external attached motor (CyberMaster Base unit only).
`Label3.Caption = PBrickCtrl.Poll 16, 0`	Label3 will be set equal to the AGC level in the receiver part of the CyberMaster Base unit.

☐ CyberMaster Command
☑ RCX Command

SelectPrgm(Number)

☐ Downloadable Command
☑ Immediate Command

Selects the active program. Used for changing active program.

This command acts like pressing the SelectPrgm button on the PBrick (RCX only).

Part:	Description:
Number:	Program number to switch to (0-4). See ParameterTable for ranges, page 10
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SelectPrgm 3
```
Selects program 4 (display shows 1-5 for program 0-4)

☑ CyberMaster Command
☑ RCX Command

StartTask(Number)

☑ Downloadable Command
☑ Immediate Command

Starts execution of PBrick's task [Number]. Tasks always start from the beginning of the task (i.e. the very first program line in the task). If the task [Number] was already running, it is stopped and then restarted (from the very first program line in the task).

If no task was running before issuing this command:

RCX:
The little man in the display on the PBrick will start running.

CyberMaster:
The Run-LED will change from flashing green to steady yellow.

Part:	Description:
Number:	The task to be started. See ParameterTable for range, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.StartTask 2
```
Task 2 (re)started and is now running.

☑ CyberMaster Command
☑ RCX Command

StopTask(Number)

☑ Downloadable Command
☑ Immediate Command

Stops execution of PBrick's task [Number]. If all tasks are stopped:

RCX:	CyberMaster:
The little man in the display on the PBrick will stop running.	The yellow Run-LED will switch to steady green (i.e. OFF).

Part:	Description:
Number:	The task to be stopped. See ParameterTable for range, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.StopTask 2
```
Task 2 stopped.

☑ CyberMaster Command
☑ RCX Command

StopAllTasks( )

☑ Downloadable Command
☑ Immediate Command

Stops execution of all the PBrick's Tasks.

RCX:
The little man in the display on the PBrick will stop running.

CyberMaster:
The Run-LED will switch off (from yellow to green).

Part:	Description:
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
Label1.Caption = PBrickCtrl.StopAllTasks
```

All tasks stopped ("Emergency stop")
If PBrick in range and the command received and carried out - the text shown in Label1 will display TRUE, if not successful the text will be FALSE.

☑ CyberMaster Command
☑ RCX Command

DeleteTask(Number)

☐ Downloadable Command
☑ Immediate Command

Deletes the addressed task [Number] in the selected program in the PBrick (If the task is running, it is stopped first). If it was the only running task:

RCX:
The little man in the display on the PBrick will stop running.

CyberMaster:
The Run-LED will switch off (yellow to solid green).

Part:	Description:
Number:	Address of task to be deleted. See ParameterTable for range, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
Label1.Caption = PBrickCtrl.DeleteTask 3
```

If the task 3 is deleted OK, the Label1 will hold the text TRUE. If no success the Label1 will hold the text FALSE.

☑ CyberMaster Command
☑ RCX Command

DeleteAllTasks( )

☐ Downloadable Command
☑ Immediate Command

Deletes all tasks of the currently selected program:

RCX:
The little man in the display on the PBrick will stop running.

CyberMaster:
All tasks are deleted and the Run-LED will switch off (yellow to green).

Part:	Description:
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SelectPrgm 2
PBrickCtrl.DeleteAllTasks
```

Select program 2 in the RCX Pbrick
Erase all tasks in program 2.

☑ CyberMaster Command
☑ RCX Command

DeleteSub(Number)

☐ Downloadable Command
☑ Immediate Command

Deletes the content of Subroutine [Number].
Any task curently using this subroutine, will automatically be stopped.

Part:	Description:
Number:	The Subroutine to be deleted. See ParameterTable for range, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

	In this program task 2 uses Subroutine 2, 3 and 4 task 3 uses Subroutine 2 and 4
`PBrickCtrl.DeleteSub 3`	Subroutine 3 is deleted and task 2 is stopped. Task 3 continues running.

☑ CyberMaster Command
☑ RCX Command

DeleteAllSubs( )

☐ Downloadable Command
☑ Immediate Command

Deletes the content of all Subroutines in the currently selected program (In CyberMaster all Subroutines).
All tasks for the selected program currently using subroutines are automatically stopped.

Part:	Description:
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

`PBrickCtrl.DeleteAllSubs`	All the memory space for subroutines is cleared.

☑ CyberMaster Command
☑ RCX Command

On(MotorList)

☑ Downloadable Command
☑ Immediate Command

Starts the motors in the list. All other properties for the motors are not affected (Power, Direction, etc.).

All motors (in MotorList) are started simultaneously.

Part:	Description:
MotorList:	An ASCII string containing the names of motors to be started. Valid names: ‘0’, ‘1’ and ‘2’. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor2, Output1 etc. See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.On “02”
```
Motor 0 and motor 2 are set on (started).

☑ CyberMaster Command
☑ RCX Command

Off(MotorList)

☑ Downloadable Command
☑ Immediate Command

This command stops the motors in the MotorList. The outputs are turned off in brake mode. All other properties for the motors are not affected (Power, Direction etc.).

All motors (in MotorList) are stopped simultaneously.

Part:	Description:
MotorList:	An ASCII string containing the names of motors to be started. Valid names: '0', '1' and '2'. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor2, Output1 etc. See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.Off  "12"
```
Motor 1 and Motor 2 are stopped in brake mode.

☑ CyberMaster Command
☑ RCX Command

Float(MotorList)

☑ Downloadable Command
☑ Immediate Command

This command turns the motor(s) in the list off in float mode i.e. the motors are stopped in a free running mode. All other properties for the motors are not affected (Power, Direction etc.).

All motors (in MotorList) are stopped simultaneously.

Part:	Description:
MotorList:	An ASCII string containing the names of motors to be stopped in floating mode. Valid names: '0', '1' and '2'. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor2, Output1 etc. See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

`PBrickCtrl.Float "02"`	The motors 0 and 2 are stopped in float mode.

☑ CyberMaster Command
☑ RCX Command

SetFwd(MotorList)

☑ Downloadable Command
☑ Immediate Command

This command sets the Direction property for the motor(s) in the list. All other properties for the motors are not affected (Power, On/Off etc.).

Direction for all motors (in MotorList) are changed simultaneously.

Part:	Description:
MotorList:	An ASCII string containing the names of motors whose Direction Property should be set to Forward. Valid names: '0', '1' and '2'. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor2, Output1 etc. See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SetFwd "0"
```

The direction of Motor 0 is set to forward.

☑ CyberMaster Command
☑ RCX Command

SetRwd(MotorList)

☑ Downloadable Command
☑ Immediate Command

This command sets the Direction property for the motor(s) in the list. All other properties for the motors are not affected (Power, On/Off etc.).

Direction for all motors (in MotorList) are changed simultaneously.

Part:	Description:
MotorList:	An ASCII string containing the names of motors which should have their direction set to Reverse. Valid names: '0', '1' and '2'. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor2, Output1 etc. See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

`PBrickCtrl.SetRwd "0"`	The Motor 0's direction is set to reverse direction.

☑ CyberMaster Command
☑ RCX Command

AlterDir(MotorList)

☑ Downloadable Command
☑ Immediate Command

This command alters the Direction property for the motor(s) in the list. All other properties for the motors are not affected (Power, On/Off, etc.).

Direction for all motors (in MotorList) are changed simultaneously.

Part:	Description:
MotorList:	An ASCII string containing the names of motors whose direction should be altered. Valid names: '0', '1' and '2'. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor2, Output1 etc. See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example: Motor 1 is running in forward direction.

`PBrickCtrl.AlterDir "1"`	The direction of Motor 1 is altered. Now Motor 1 is running in the other direction.

☑ CyberMaster Command
☑ RCX Command

SetPower(MotorList, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Sets the Power property for the motor(s) in the MotorList. All other properties for the motors are not affected (On/Off, Direction etc.).

Power for all motors (in MotorList) are changed simultaneously.

Part:	Description:
MotorList:	An ASCII string containing the names of motors to be started. Valid names: '0', '1' and '2'. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor2, Output1 etc. See ParameterTable for ranges, page 9.
Source, Number:	Addresses the type and number of the source for the power level setting. See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SetPower( "012", 0, 15 )
```

The power of Motor 0, 1 and 2 is set to the power level stored in Variable 15.

☑ CyberMaster Command
☐ RCX Command

Drive(Number0, Number1)

☐ Downloadable Command
☑ Immediate Command

This command sets all the properties for both motor 0, and motor 1.
This command is useful when fast updating of the driving motors is needed. E.g. when controlling the PBrick directly by a joystick.

(The same function could have been achieved by sending these commands:
On/off+SetPower+SetFwd/SetRwd).

Part:	Description:
Number0, Number1:	Data for motor 0 (left motor) and motor 1 (right motor). Negative numbers means Rwd, positive numbers means Fwd. Zero means stop (in brake mode). See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

`PBrickCtrl.Drive( -7, 7 )`	The left motor set on, full speed and backwards. The right motor set on, full and forward, i.e. the model turns on the spot.

☑ CyberMaster Command
☐ RCX Command

OnWait(Motorlist, Number, Time)

☑ Downloadable Command
☐ Immediate Command

This command is used to start one or more motor(s) with a specified power level and then wait for a specified time. When the wait time is finished, the user application decides what to do next (continue or stop etc.).
If the command is used with a time setting of 0, the command will act as a normal ON, but with an additional power level/direction specified.

Part:	Description:
Motorlist:	An ASCII string containing the names of motors to be started. Valid names: '0', '1' and '2'. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor2, Output1 etc.
Number:	Negative "Number" means reverse direction, positive "Number" means forward direction.. Zero means stop (in brake mode).
Time:	The wait "Time" can be set to 0- 255, the time is in counts of 100ms. (I.e. 0 to 25.5 sec.). A zero (0) means no wait (i.e. the command acts as a normal On/Off command). See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

`PBrickCtrl.OnWait "12", -3, 25`	Motor 1 and 2 On, Reverse direction, Power Level 3 and Wait 2.5 sec.
`PBrickCtrl.OnWait "12", 5, 0`	Command used as an On command with additional power setting.

☑ CyberMaster Command
☐ RCX Command

OnWaitDifferent(Motorlist, Number0, Number1, Number2, Time)

☑ Downloadable Command
☐ Immediate Command

This command is used to start more motors with different power levels/directions and then wait for a specified time. When the wait time is finished, the user application decides what to do next (continue or stop etc.).
If the command is used with a time setting of 0, the command will act as a normal ON, but with an additional power level/direction specified.

Part:	Description:
Motorlist:	An ASCII string containing the names of motors to be started. Valid names: '0', '1' and '2'. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor2, Output1 etc.
Number0, Number1, Number2:	Negative "NumberX" means reverse direction, positive "NumberX" Means forward direction.. Zero means stop (in brake mode).
Time:	The wait "Time" can be set to 0- 255, the time is in counts of 100ms (i.e. 0 to 25.5 sec.) A zero (0) means no wait (i.e. the command acts as a normal On/Off command). See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.OnWaitDifferent "012", -3, 3, -7, 25
```

Motor 0 On Power level 3 Reverse direction.
Motor 1 On Power level 3 Forward direction.
Motor 2 On Power level 7 Reverse direction.
Wait 2.5 sec.

☑ CyberMaster Command
☐ RCX Command

ClearTachoCounter(MotorList)

☑ Downloadable Command
☑ Immediate Command

This command clears the TachoCounter. The TachoCounter is an integrated part of the 2 internal motors.

Part:	Description:
MotorList	An ASCII string containing the names of motors to be started. Valid names: '0' and '1'. But the ActiveX control will search the string and remove other characters, so more readable names can be used: E.g. Motor0 etc. See ParameterTable for ranges, page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.ClearTachoCounter "1"
```
Clears the Tachovalue of Motor 1.

☑ CyberMaster Command
☑ RCX Command

PlayTone(Frequency, Time)

☑ Downloadable Command
☑ Immediate Command

This command is used to make the PBrick play a tone via the internal speaker. The Frequency parameter sets the pitch of the tone and the Time parameter sets the duration of the tone. The tones will be buffered in the RCX, if more than one tone is sent.

Part: **Description:**

Frequency: Sets the frequency of the Tone. All integers in the range can be selected.
For range: See ParameterTable page 10.

To help those who wants to play 'music', there is a table below containing the frequencies for the notes in eight octaves. The frequencies are rounded to integers. (C4 is the middle C).

PITCH	1	2	3	4	5	6	7	8
G#	52	104	208	415	831	1661	3322	
G	49	98	196	392	784	1568	3136	
F#	46	92	185	370	740	1480	2960	
F	44	87	175	349	698	1397	2794	
E	41	82	165	330	659	1319	2637	
D#	39	78	156	311	622	1245	2489	
D	37	73	147	294	587	1175	2349	
C#	35	69	139	277	554	1109	2217	
C	33	65	131	262	523	1047	2093	4186
B	31	62	123	247	494	988	1976	3951
A#	29	58	117	233	466	932	1865	3729
A	28	55	110	220	440	880	1760	3520

Time: The duration of the Tone, in 10ms steps.
For range: See ParameterTable page 10.

Return value: If the function succeeds, the return value is TRUE.
If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.PlayTone 2000, 100
```
Plays 2000 Hz. for a duration of 1 sec.

☑ CyberMaster Command
☑ RCX Command

PlaySystemSound(Number)

☑ Downloadable Command
☑ Immediate Command

This command is used to make the RCX play one of 6 pre-defined sound patterns. The sounds will be buffered in the RCX, if more than one is sent.

Part:	Description:

Number: Number addresses the sound to play.
Below is a short description of the predefined sound patterns, and some guidelines for using them:

Number	Sound	Purpose
0	'Key click'	Used by default when a key is pressed.
1	'Beep beep'	Normally used as an 'acknowledge'.
2	Decreasing frequency sweep	Used to indicate end of successful download.
3	Increasing frequency sweep	Used by default to indicate end of successful upload (e.g. Datalog)
4	'Buhhh'	Error sound
5	Fast increasing sweep	Used three times in a row to indicate: 'Hurrah sound'

Return value: If the function succeeds, the return value is TRUE.
If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.PlaySystemSound 4
```
Audibly flag an error

☐ CyberMaster Command
☑ RCX Command

SetSensorType(Number, Type)

☑ Downloadable Command
☑ Immediate Command

This command is used to specify the SensorType for an input. The Type information (and SensorMode) tells the RCX how to use and represent Sensor data.
For information about SetSensorMode see page 69. See also the ‘Inputs’ on page 100.

Part:	Description:
Number:	Addresses the input port for which the Type has to be set.
Type:	Specifies the SensorType: 0: None 1: Switch 2: Temperature 3: Reflection (Light sensor) 4: Angle
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SetSensorType 0, 4
```
Input 0 setup for using an Angle sensor.

☑ CyberMaster Command
☑ RCX Command

SetSensorMode(Number, Mode, Slope)

☑ Downloadable Command
☑ Immediate Command

This command sets the SensorMode of the input. The analogue values can be represented and calibrated in different pre-defined values. The digital representation can also be set to different modes. By adding dynamic measurements, the user can apply a sort of “High Pass”-filter. The change on the input shall have a certain minimum dVoltage/dTime. The mode information works closely together with the SensorType setting.
For information about SetSensorType see page 68. See also the ‘Inputs’ on page 100.

Part:	Description:

Number: The input for which the Mode has to be set.

Mode: The sensor mode:

0:	**Raw**	Raw analogue data (0-1023)
1:	**Boolean**	TRUE/FALSE
2:	**Transition Counter**	All transitions are counted (both positive and negative transitions are counted).
3:	**Periodic Counter**	Only counting whole periods (one negative edge + a positive edge – or ‘vice versa).
4:	**Percent**	Sensor value represented in percent of full scale.
5:	**Celsius**	Measurement represented in Celsius.
6:	**Fahrenheit**	Measurement represented in Fahrenheit.
7:	**Angle**	Input data counted as Angle steps.

Slope: If Boolean mode of operation is selected, Slope indicates how to determine TRUE and FALSE in SensorValue. This also affects the way counters reacts on input changes.

0:	Absolute measurement (below 45% of full scale = TRUE, above 55% of full scale = FALSE). i.e. a pushed switch (low voltage measured) results in a TRUE state.
1-31:	Dynamic measurement. The number indicates the size of the dynamic slope. I.e. the necessary change of bit-counts between two samples, to get a change in the Boolean state.

Return value: If the function succeeds, the return value is TRUE.
If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SetSensorType 1, 1
PBrickCtrl.SetSensorMode 1, 1, 0
```

The sensor at input 1 is set as a switch
The switch data should be represented as boolean and an absolute value.

☑ CyberMaster Command
☑ RCX Command

ClearSensorValue(Number)

☑ Downloadable Command
☑ Immediate Command

This command clears the sensor value register. (On the RCX: The sensor value register holds the value which is shown in the display).

Part:	Description:
Number:	Addresses the input whose input value register should be cleared. See ParameterList for ranges on page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

	An angle sensor is connected to input 1 of an RCX. The angle sensor is turned 360 degrees 3 times and the display shows 48 (3 * 16) steps.
`PBrickCtrl.ClearSensorValue 1`	Clear the sensor value register for input 1 (I.e. the middle one labelled 2 on the RCX). Now the display on the RCX shows an angle sensor reading of zero (0).

☑ CyberMaster Command
☑ RCX Command

ClearTimer(Number)

☑ Downloadable Command
☑ Immediate Command

This command clears one of the four free-running Timers. The Number parameter indicates which timer to clear. After this command is executed, the timer is set to zero (0), and the timer is restarted.

Part:	Description:
Number:	Indicates which timer to clear. For information about ranges see ParameterTable on page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

`PBrickCtrl.ClearTimer 2`	Clears Timer 2 and restarts it.

☑ CyberMaster Command
☑ RCX Command

GoSub(Number)

☑ Downloadable Command
☐ Immediate Command

This command is only for download. A subroutine cannot call another subroutine since the PBrick has no call stack, but only a single return pointer. If the subroutine does not exist the call is ignored. For information about how to download a subroutine see the function BeginOfSub on page 42.

Part:	Description:
Number:	Addresses the subroutine to call. For information about range etc. see ParameterTable on page 10.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.BeginOfSub 1                 Starts Subroutine 1
  PBrickCtrl.PlaySystemSound 2          Makes PBrick play SystemSound 2
  PBrickCtrl.On “12”                    Starts motors at output 1 and 2
PBrickCtrl.EndOfSub                     Ends Subroutine 1 declaration


PBrickCtrl.BeginOfTask 0                Starts definition of Task 0
  PBrickCtrl.GoSub 1                    1st call of Subroutine 1
  ...
  PBrickCtrl.GoSub 1                    2nd call of Sub 1
PBrickCtrl.EndOfTask                    End of Task 0
```

☑ CyberMaster Command
☑ RCX Command

Loop(Source, Number)

☑ Downloadable Command
☐ Immediate Command

This command is part on the Loop…EndLoop control structure.
Program lines between Loop and EndLoop will be repeated as many times as the Source/Number states at runtime.
However, there is an exception to this rule: A Loop 2, 0 statement means infinite loop.
If a Loop zero (0) is set-up via a variable or a random number at runtime, the Loop-EndLoop structure will be entirely skipped, so the execution will not end in a deadlock if the source becomes zero (0).

Part:	Description:
Source, Number:	Addresses and type for the source of the loop value. See ParameterTable on page 9 for information about range.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example 1:

```
PBrickCtrl.Loop 0, 3
  ...



PBrickCtrl.EndLoop
```

Get the Loop count from variable 3
If variable 3 is zero (0) when the Loop 0, 3 statement is entered first time, this line and the following lines are skipped.
If variable 3 was 2, when the Loop 0, 3 was entered first time, these lines are executed 2 times.

Example 2:

```
PBrickCtrl.Loop 2, 0
  ...
PBrickCtrl.EndLoop
```

Loop Forever

☑ CyberMaster Command
☑ RCX Command

EndLoop( )

☑ Downloadable Command
☐ Immediate Command

This command terminates the Loop…EndLoop control structure. For information about Loop see page 74).

The EndLoop causes the loop count to be decremented and tested. If the loop count has not reached zero (0), the program-execution is repeated again from the corresponding Loop statement.
If a Loop 2, 0 is used, the EndLoop acts as a normal unconditional jump back to the beginning of the loop (i.e. loop forever).

Part:	Description:
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.Loop 2, 0
  ...
PBrickCtrl.EndLoop
```

Loop forever.

Jumps back to loop start and do it “forever”.

☑ CyberMaster Command
☑ RCX Command

While(Source1, Number1, RelOp, Source2, Number2)

☑ Downloadable Command
☐ Immediate Command

This command is a part of the While…EndWhile control structure.
The program lines located between the While and EndWhile statement will be executed as long as the Condition described by parameters evaluates TRUE.
The two values tested are addressed by Source1/Number1 and Source2/Number2.

Part:	Description:
Source1, Number1:	Addresses the first CompareValue. Check the ParameterTable on page 9 for types and ranges.
RelOp:	This specifies the relational operator used for the compare of the two CompareValues.

Relation Operator:	Constant equivalent:
>	0
<	1
=	2
< >	3

Source2, Number2:	Addresses the second CompareValue. See ParameterTable on page 9 for information about types and ranges.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SetVar 5, 2, 0
PBrickCtrl.While 0, 5, 1, 2, 10
  PBrickCtrl.PlayTone 1000, 10
  PBrickCtrl.Wait 100
  PBrickCtrl.SumVar 5, 2, 1
PBrickCtrl.EndWhile
```

Sets variable 5 = 0
While variable 5 < (constant) 10
Plays a 1000Hz tone for a 100ms.
Waits 1 sec.
Increment variable 5 by 1.
Ends the While…EndWhile control structure.
The tone is played 10 times, with a 1 sec. interval.

☑ CyberMaster Command
☑ RCX Command

EndWhile( )

☑ Downloadable Command
☐ Immediate Command

This command terminates the While…EndWhile control structure.
For further information see the While statement on page 76.

Part:	Description:
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
  ...
PBrickCtrl.While 0, 0, 0, 2, 3
  ...
PBrickCtrl.EndWhile
```

While variable 0 > (constant) 3

☑ CyberMaster Command
☑ RCX Command

If(Source1, Number1, RelOp, Source2, Number2)

☑ Downloadable Command
☐ Immediate Command

This command is a part of the If…[Else]…EndIf control structure.
The program lines located between the If and EndIf statements will be executed if the condition described by the parameters evaluates to TRUE.
If an Else block exists, this block will be executed if the If-statement evaluates to FALSE.
The two tested values are addressed by Source1/Number1 and Source2/Number2.

Part:	Description:
Source1, Number1:	Addresses the source and type of the first compare value for the Compare. See the ParameterTable on page 9 for ranges and type.
RelOp:	This specifies the relational operator used for the compare of the two Compare values.

Relation Operator:	Constant equivalent:
>	0
<	1
=	2
< >	3

Source2, Number2:	Addresses the source and the type for the second compare value. See ParameterTable on page 9 for ranges and type.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.If 0, 5, 1, 2, 10
  PBrickCtrl.PlayTone 1000, 10
PBrickCtrl.EndIf( )

PBrickCtrl.If 0, 5, 1, 2, 8
  PBrickCtrl.PlayTone 1000, 10
PBrickCtrl.Else
  PBrickCtrl.PBTurnOff
PBrickCtrl.EndIf
```

If variable 5 < (constant)10 then
play a 1000 Hz tone for 100ms.
Close the conditional If…EndIf control structure.

If variable 5 < (constant) 8 then
play a 1000 Hz tone for 100ms.
Start of the Else part
Else If variable 5 >= 8 then turn the PBrick OFF
Close the If…Else…EndIf control structure.

☑ CyberMaster Command
☑ RCX Command

Else( )

☑ Downloadable Command
☐ Immediate Command

This command is part of the If…Else...EndIf control-structure.
See the If command on page 78 for further information.

Part:	Description:
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.If 2, 5, 2, 0, 15
  PBrickCtrl.PlayTone 440, 20
PBrickCtrl.Else
  PBrickCtrl.PlayTone 5000, 10
PBrickCtrl.EndIf
```

If (constant) 5 = variable 15 then
play a 440 Hz tone for 200 ms.
Else
play a 5000 Hz. tone with a 100 ms. duration
Close the If...Else...EndIf control-structure.

☑ CyberMaster Command
☑ RCX Command

EndIf( )

☑ Downloadable Command
☐ Immediate Command

This command is part of the If…[Else]...EndIf control structure. It closes the conditional control structure.
For more information about the If statement see page 78.

Part:	Description:
Return value:	If the command/function succeeds, the return value is TRUE. If the command/function fails, the return value is FALSE.

Example:

```
PBrickCtrl.If 0, 0, 1, 0, 2
  PBrickCtrl.On “01”
PBrickCtrl.EndIf
```

If variable 0 < variable 2 then
Turns motor 0 and motor 1 ON
Closes the conditional part of program execution

☑ CyberMaster Command
☑ RCX Command

Wait(Source, Number)

☑ Downloadable Command
☐ Immediate Command

This command is used to stop the program execution. It is only execution in the calling task which is suspended for some time. The program execution is exclusively handed over to the other tasks.

Part:	Description:
Source, Number:	Addresses the source and the type for the waiting time [10 ms. resolution]. See the ParameterTable on page 9 for information about for ranges and type.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.On “1”
PBrickCtrl.Wait 2, 1000
```

Turns output 1 ON
Waits (suspend execution in this task) for 10 Sec.

☑ CyberMaster Command
☑ RCX Command

SetVar(VarNo, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Sets the [VarNo] variable to the value addressed by the Source and Number parameters.

Part:	Description:
VarNo:	The variable number to be set.
Source, Number:	Addresses the type and the source of the new value for the variable. See the ParameterTable on page 9 for more information about range and type.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SetVar 16, 2, 33
```

Initialises Variable 16 with the constant value 33.

☑ CyberMaster Command
☑ RCX Command

SumVar(VarNo, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Adds the value addressed by Source and Number to the [VarNo] variable. Result is stored in the [VarNo] variable.

Part:	Description:
VarNo:	The [VarNo] variable is both source of the value to be added and also the destination for the result of the addition.
Source, Number:	This parameter addresses the source and type of the second source. See the ParameterTable on page 9 for information about ranges of source and number.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SumVar 0, 0, 0
```
Adds Variable 0 to itself (Variable 0 = 2 * Variable 0).

☑ CyberMaster Command
☑ RCX Command

SubVar(VarNo, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Subtracts the value addressed by Source and Number from the [VarNo] variable.

Part:	Description:
VarNo:	The [VarNo] variable is both source of the value to be subtracted and also the destination for the result of the subtraction.
Source, Number:	Addresses the value to subtract from the [VarNo] variable. See ParameterTable on page 9 for range.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SubVar 0, 2, 1
```
Decrements Variable 0 by 1

☑ CyberMaster Command
☑ RCX Command

DivVar(VarNo, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Divides the [VarNo] variable with the value addressed by Source and Number. The result of the division is stored in the [VarNo] variable.
The result is always rounded down to the nearest integer.
If the division results in a “divide by zero”, the result of the operation is defined and set to zero (0).

Part:	Description:
VarNo:	The [VarNo] variable is both source of the value to be divided and also the destination for the result of the division.
Source, Number:	Addresses the divisor. See ParameterTable on page 9 for ranges.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

`PBrickCtrl.DivVar 0, 2, 3`	Divides Variable 0 by 3. If Variable 0 is holding the value of 5 then the result will be 1 (always rounded down).

☑ CyberMaster Command
☑ RCX Command

MulVar(VarNo, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Multiplies the [VarNo] variable with the value addressed by Source and Number. The result of the multiplication is stored in the [VarNo] variable.
If the result is bigger than a signed 16 bit integer, the result is rounded to lie within the interval: -32768 or 32767.

Part:	Description:
VarNo:	The [VarNo] variable is both source of the value to be multiplied and also the destination for the result of the multiplication.
Source, Number:	Addresses the value to multiply with the [VarNo] variable. See ParameterTable on page 9 for ranges.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.MulVar 2, 2, 8
```
Multiplies Variable 2 by a constant of 8.

☑ CyberMaster Command
☑ RCX Command

SgnVar(VarNo, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Stores the result of the test of the value addressed by Source and Number in the [VarNo] variable.

If the addressed value > **0** then [VarNo] variable is set to **1.**
If the addressed value = **0** then [VarNo] variable is set to **0.**
If the addressed value < **0** then [VarNo] variable is set to **-1**.

Part:	**Description:**
VarNo:	Addresses the [VarNo] variable to hold the result of the sign test.
Source, Number:	Addresses the source for the sign test. See ParameterTable on page 10 for ranges.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.SgnVar 0, 0, 12
```

If variable 12 = -24 then Variable 0 is set to -1
If variable 12 = 0 then Variable 0 is set to 0
If variable 12 = 2255 then Variable 0 is set to 1

☑ CyberMaster Command
☑ RCX Command

AbsVar(VarNo, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Stores the absolute value of the value addressed by Source and Number in the [VarNo] variable.

Part:	Description:
VarNo:	The [VarNo] variable used as destination for the result.
Source, Number:	Addresses the source and type for the requested value, from which the Abs-value should be evaluated. Look in the ParameterTable on page 9 for range.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.AbsVar 2, 0, 7
```
If variable 7 is -33 then variable 2 is set to 33 (Abs of -33)

☑ CyberMaster Command
☑ RCX Command

AndVar(VarNo, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Performs a bitwise AND operation between [VarNo] variable and the value addressed by Source and Number. The result is stored in the [VarNo] variable.

Part:	Description:
VarNo:	The [VarNo] variable is both source of the value to be AND'ed and also the destination for the result of the AND operation.
Source, Number:	Addresses the source and type of the second source for the AND. See the ParameterTable on page 10 for ranges.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.AndVar 0, 2, 7
```

If Variable 0 holds the value 17 decimal (10001 binary) then this command line will put the value 1 into Variable 0.

☑ CyberMaster Command
☑ RCX Command

OrVar(VarNo, Source, Number)

☑ Downloadable Command
☑ Immediate Command

Performs a bitwise OR operation between [VarNo] variable and the value addressed by Source and Number. The result is stored in the [VarNo] variable.

Part:	Description:
VarNo:	The [VarNo] variable is both source of the value to be OR'ed and also the destination for the result of the OR operation.
Source, Number:	Addresses the source and type for the bitwise OR operation. See the ParameterTable on page 9 for ranges.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.OrVar 5, 2, 1
```

If Variable 5 is set to 4 (100 binary)
then this command will fill 5 into Variable 5 (101 binary).

☐ CyberMaster Command
☑ RCX Command

SetDatalog(Size)

☐ Downloadable Command
☑ Immediate Command

Allocates the datalog area. A previous allocated datalog area is automatically erased. Each element of a datalog allocates 3 bytes in the PBrick. The datalog area ranges from 1 to Size. The element 0 of the datalog area (can be accessed by the UploadDatalog(0, 1)) always reflects the maximum available datalog area [Size].

Part:	Description:
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

`PBrickCtrl.SetDatalog 50`	Initialises a datalog area of 50 elements.

☐ CyberMaster Command
☑ RCX Command

DatalogNext(Source, Number)

☑ Downloadable Command
☑ Immediate Command

This command forces a new sample of the value addressed by the Source and Number. The PBrick automatically increments its internal datalog pointer. If the end of the datalog area is reached, nothing happens. I.e. the Datalog area is **not** overwritten by automatic "wrap around".

The user can use a counter in the RCX and use SetDatalog(Size) from a PC program monitoring the counter to make a pseudo wrap around.

Part: **Description:**

Source, Number:

Value to log:	Source:	Number:
Var 0 - 31	0	0 - 31
Timer 0 - 3	1	0 - 3
$Input_{(SensorValue)}$ 0 - 2	9	0 - 2
Watch	14	0

Return value: If the function succeeds, the return value is TRUE.
If the function fails, the return value is FALSE.

Example:

	Asynchronous datalog. The individual samples are time stamped.
`PBrickCtrl.DatalogNext 14, 0`	Time stamp
`PBrickCtrl.DatalogNext 9, 1`	Datalog sensor 1
`PBrickCtrl.DatalogNext 9, 0`	Synchronous datalog. E.g. the sampling forced by a timer or another trigger.

☐ CyberMaster Command
☑ RCX Command

UploadDatalog(From, Size)

☐ Downloadable Command
☑ Immediate Command

This command is used for getting the logged data from the RCX. The start and size of the uploaded datalog is defined in From and Size. For information about SetDatalog see page 91.

Part:	Description:
From:	Addresses the start point for the upload.
Size:	Defines the size of the upload.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
Private Sub Command1_Click()
  Dim arr As Variant                                          Variant array-type
  Dim i As Integer                                            Array index stepper
  Dim from As Integer                                         "Pointer" to first wanted element
  Dim datalength As Integer                                   Size of upload

  from = Val(Text1.Text)                                      User request of starting point
  datalength = Val(Text2.Text)                                Size of requested upload

  arr = PBrickCtrl.UploadDatalog(from, Datalength)            Perform the Upload

  If IsArray(arr) Then                                        Check for a valid array (variant)

     For i = LBound(arr, 2) To UBound(arr, 2)                 Iterate over the whole array

       List1.AddItem "Type: " + Str(arr(0, i)) +              Display the uploaded data in the
                     "  No. " + Str(arr(1, i)) +              Listbox
                     "  Value: " + Str(arr(2, i))

                                                             "List1":
                                                             Type: x   No. y   Value: zzzz
     Next I                                                  Continue until all elements are
                                                             displayed
  Else                                                       If arr not a valid array
     MsgBox "Upload NOT a valid array"                       Some debug info
  End If
End Sub                                                      End of upload
```

☐ CyberMaster Command
☑ RCX Command

SendPBMessage(Source, Number)

☑ Downloadable Command
☐ Immediate Command

This command makes the PBrick transmit a message on the IR-communication channel.
The command enables two PBrick's to communicate/interact with each other, without any link to a PC. It is **not** possible to use this command and having a PC communicating with the PBrick simultaneously. This is due to the IR-channel being common to both sorts of communication.

Part:	Description:
Source, Number:	Addresses the type and source for use as message. For ranges see ParameterTable on page 9.
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.BeginOfTask 2
   PBrickCtrl.If 0, 2, 2, 2, 55
     PBrickCtrl.SendPBMessage 0, 10
   PBrickCtrl.EndIf
PBrickCtrl.EndOfTask
```

If Variable 2 = 55
then send the PBrick message stored in Variable 10 to the other PBrick.

☐ CyberMaster Command
☑ RCX Command

ClearPBMessage( )

☑ Downloadable Command
☐ Immediate Command

Clears the PBrick message stored internally in the PBrick.

Part:	Description:
Return value:	If the function succeeds, the return value is TRUE. If the function fails, the return value is FALSE.

Example:

```
PBrickCtrl.BeginOfTask 3
   PBrickCtrl.ClearPBMessage
PBrickCtrl.EndOfTask
```

Clears the PBrick message register.

☑ CyberMaster Command
☑ RCX Command

[OLE Event]: VariableChange (Number, Value)

This is the event sent from the ActiveX control when the addressed source has changed value.
By using this function the impact on the user application will be minimal, when constantly checking resources in the PBrick.
Currently only changes of variable zero (0) can be polled this way. The user can download tasks which automatically update variable 0 with different data.
The source, type and address are set by the command SetEvent, see page 37.

Part:	Description:
Number:	The address of the autopolled and changed data.
Value:	The value of the autopolled and changed data.

Example:

```
Private Sub Command1_Click()
  Dim Src As Integer                                  Type of source for the event

  Dim No As Integer                                   Address of the source
  Dim Tim As Integer                                  Autopoll time

  Src = Val(Text1.Text)                               User input for what to poll
  No = Val(Text2.Text)                                and how often.
  Tim = Val(Text3.Text)
  Label1.Caption = PBrickCtrl.SetEvent(Src, No, Tim)  Setup the autopoll
End Sub


Private Sub PBrickCtrl_VariableChange(ByVal Number As Integer,
                                      ByVal Value As Integer)
  Label2.Caption = Value                              Show the autopolled data
End Sub
```

☑ CyberMaster Command
☑ RCX Command

[OLE Event]: DownloadDone (ErrorCode, TaskNo)

The DownloadDone event is sent from the ActiveX control as soon as the download is finished or an error has terminated the download.
ErrrorCode is an error flag. The TaskNo addresses which tasknumber or subroutine number the error flag refers to.
If the download is a firmware download (RCX only) the TaskNo will always contain the number 100.
Use the AsyncronBrickError event, if ErrorCode <> 0 (i.e. ErrorCode = 1), to get more information about an error.

Part: **Description:**

ErrorCode:

ErrorCode [name]:	ErrorCode [number]:	
	RCX:	CyberMaster:
OKDownload.	0x00	0x00
Download Failed. The user should use the AsyncronBrick error to get more information.	0x01	0x01

The variable TaskNo can signal: Task numbers, sub numbers and firmware.

TaskNo.:

RCX:		CyberMaster:	
Task:	00-09	Task:	00-04
Sub:	10-17	Sub:	10-13
Firmware:	100	----------	

Example: (A TASK download)

```
Private Sub PBrickCtrl_DownloadDone(ByVal ErrorCode As Integer,
                                    ByVal TaskNo As Integer)

  If ErrorCode <> 0 Then

    Label1.Caption = "Error in download"

  Else
    Label1.Caption = "Download OK"
  End If
  Label2.Caption = TaskNo
End Sub
```

If a download error occurs 2 labels are set to values of ErrorCode and TaskNo respectively.

Flag error.

Flags an OK download

Reports the download number (type)

☑ CyberMaster Command
☑ RCX Command

[OLE Event]: DownloadStatus (timeInMS, sizeInBytes, taskNo)

This event is used to get an estimated download time, the size of the compiled code for download and a reference to the task or sub. If an application only needs the time for a download and/or the size of the task/sub, but not a real download, then the EndOfTaskNoDownload (or EndOfSubNoDownload) should be used.

Part:	Description:
TimeInMS	Estimated download time without any retransmissions etc. The value is returned as a 32 bit integer Long.
sizeInBytes	The size of the compiled task or subroutine. The value is returned as a Long.
taskNo	This number represents a reference to the task, sub or download of firmware. The value is returned as an Integer.

TaskNo.:

RCX:		CyberMaster:	
Task:	00-09	Task:	00-04
Sub:	10-17	Sub:	10-13
Firmware:	100	----------	

Example:

```
Private Sub PBrickCtrl_downloadStatus(ByVal timeInMS As Long,
                                      ByVal sizeInBytes As Long,
                                      ByVal taskNo As Integer)
  Label1.Caption  = timeInMS
  Label2.Caption  = sizeInBytes
  Label3.Caption  = taskNo
End Sub
```

☑ CyberMaster Command
☑ RCX Command

[OLE Event]: AsyncronBrickError (Number, Description)

This event is sent from the download thread via the ActiveX control. If the DownloadDone event returns an error code <> from zero (0), the application should use this event to get more information. See appendix A, page 102 for more information about asynchronous error handling.

Part:	Description:
Number:	A number referring to a specific error. See appendix B on page 104 for more information about the error codes.
Description:	A textual error message. These messages can e.g. be used by the application as a text in a MessageBox. See appendix B, page 104 for more information about these messages.

Example:

```
Private Sub PBrickCtrl_AsyncronBrickError(ByVal Number As Integer,
                                          Description As String)
  Label1.Caption = Number
  Label2.Caption = Description
End Sub
```

Inputs:

There are 3 inputs which are sampled by a 10bit A/D converter.

RCX:	CyberMaster:
Inputs are able to source power for active sensors. The power sourcing is switched on and off, depending on the type of sensor attached to the input.	Inputs are not able to source any power for active sensors.

Outputs:

There are 3 Outputs capable of sourcing 9V power for LEGO motors, bulbs etc.
Power can be controlled in 8 power levels. (See SetPower, page 61).

Immediate Control:

It is possible to control the PBrick without downloading any programs to it. These commands are executed immediately and are called immediate commands. See OCX Overview page 8.

Tasks:

The PBrick provides a multitasking environment, making it possible to execute up to 10 tasks in parallel (4 for CyberMaster).
All tasks have access to an interpreter, which executes the downloaded commands.

As soon as a program sequence is downloaded to one of the tasks, it is possible to start program execution of that task (See StartTask, page 48).

Events:

It is possible to set-up the ActiveX control to automatically poll the PBrick for status on Variable 0, and then generate an event if changes has occurred.
This is a time-optimised way of getting information of changes in the PBrick.

(See SetEvent, page 37).

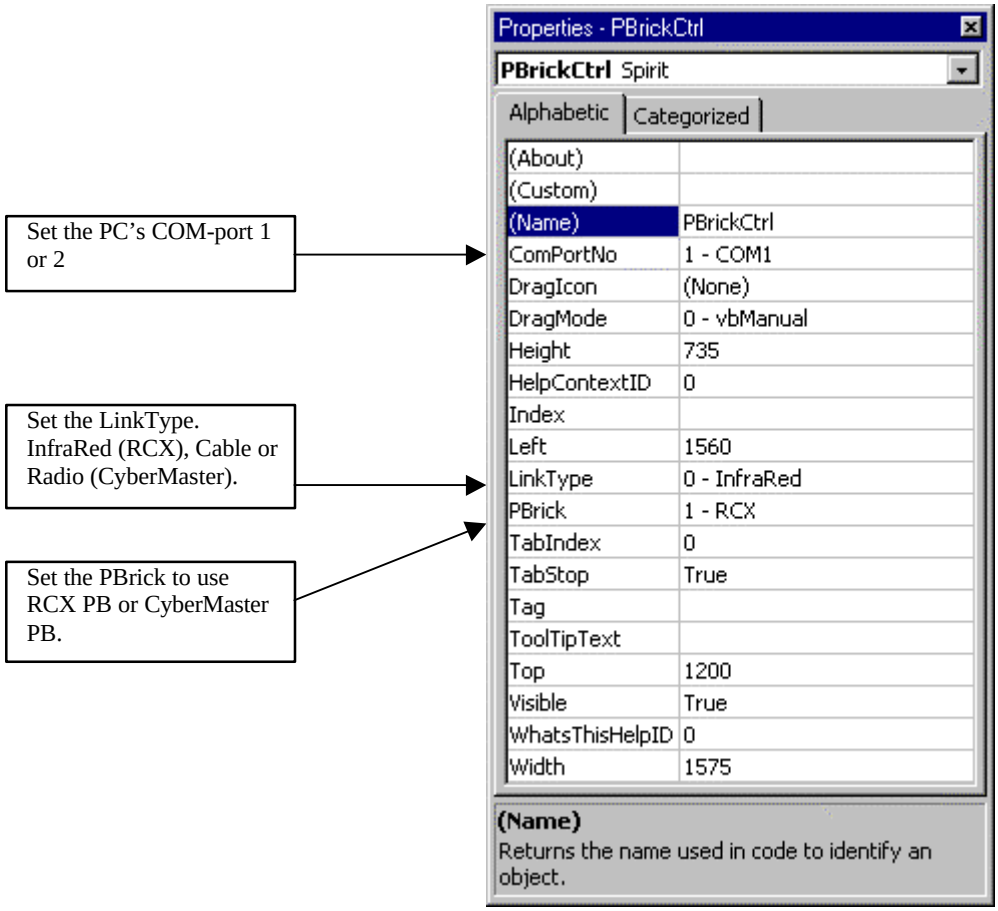
Timers:

There are 4 free-running Timers in the PBrick, with a resolution of 100 ms. They can be cleared individually. As soon as they are cleared they start running again (See ClearTimer, Page 72).

Variables:

There are 32 (Global) variables in the PBrick, defined as signed 16 bit integers within the interval: -32768 to 32767.

Properties:



Appendices

Appendix A:

Errorhandling while downloading code to the PBrick:

The DownloadDone event reports the result of the operation. If the ErrorCode returned by the DownloadDone event <> zero (0) an error has occured.

To get information about this error, it is necessary to check the AsyncronBrickError event. But it is not as simple as it appears. If the OCX (ActiveX) control sends an event and forces a dialog-box to be opened, all other events sent from the ActiveX control to the Visual Basic application will disappear.

A possible workaround is outlined below:

1) A flag should be defined in the General/Declaration section:

```
Dim waitForDD As Boolean
```

2) In the Form_Load event handler this flag is set to false (i.e. initialised as “not waiting”):

```
waitForDD = False
```

3) In the AsyncronBrickError event handler the application checks for the “waiting on a DownloadDone event” i.e. check the waitForDD flag.
By doing so, the AsyncronBrickError event handler waits until the DownloadDone event has occurred and the DownloadDone event handler has finished its job.
This means the two events are synchronised.

```
Private Sub PBrickCtrl_AsyncronBrickError(ByVal Number As Integer,
                                          Description As String)
  If (waitForDD) Then
     While (waitForDD)
        DoEvents
     Wend
     MsgBox "AsyncronBrickError: " + Str(Number) + " " + Description
  Else
     MsgBox "AsyncronBrickError: " + Str(Number) + " " + Description
  End If
End Sub
```

4) The DownloadDone event handler resets the synchronisation flag. I.e. it clears waitForDD.

```
Private Sub PBrickCtrl_DownloadDone(ByVal ErrorCode As Integer,
                                    ByVal DownloadNo As Integer)

  If ErrorCode = 0 Then
    MsgBox "Download Done and OK"      ok
  Else
    MsgBox "Download Failed!"
  End If
  waitForDD = False
End Sub
```

5) In the application the synchronisation flag waitForDD should be set before any downloading takes place.

```
-
-
waitForDD = True                       Set the flag.
-
-
PBrickCtrl.SetSensorType 0, 2          Sensor 0 is a temperature and
PBrickCtrl.SetSensorMode 0, 5, 0       configured to read Celsius

PBrickCtrl.BeginOfTask 1               Begin task
  PBrickCtrl.Loop 2, 0
    PBrickCtrl.SetVar 0, 9, 0
    PBrickCtrl.Wait 2, 90
  PBrickCtrl.EndLoop
PBrickCtrl.EndOfTask                   End task. I.e. download the task.
-
-
```

Appendix B:

LEGOPBRICK ERRORCODES:

"UNKNOWN"		
	UNKNOWN	= 2000
"FATAL"		
	FATAL	= 2001
"PROGRAM_ERROR"		
	PROGRAM_ERROR	= 2002
"PIPE_TO_THREAD_ERROR_SENDER"		
	PIPE_TO_THREAD_ERROR_SENDER	= 2003
"PIPE_TO_THREAD_ERROR_RECEIVER"		
	PIPE_TO_THREAD_ERROR_RECEIVER	= 2004
"TOO_MANY_RESENDS"		
	TOO_MANY_RESENDS	= 2005

Download Thread, syntax errors

"SYNTAX_MATCHING_ENDIF_NOT_FOUND"		
	SYNTAX_MATCHING_ENDIF_NOT_FOUND	= 2006
"SYNTAX_MATCHING_ENDWHILE_NOT_FOUND"		
	SYNTAX_MATCHING_ENDWHILE_NOT_FOUND	= 2007
"SYNTAX_MATCHING_ENDLOOP_NOT_FOUND"		
	SYNTAX_MATCHING_ENDLOOP_NOT_FOUND	= 2008
"SYNTAX_END_REACHED_TOO_SOON"		
	SYNTAX_END_REACHED_TOO_SOON	= 2009
"SYNTAX_TO_MANY_NESTED_LOOPS_IN_TASK"		
	SYNTAX_TO_MANY_NESTED_LOOPS_IN_TASK	= 2010
"SYNTAX_TO_MANY_NESTED_LOOPS_IN_SUB"		
	SYNTAX_TO_MANY_NESTED_LOOPS_IN_SUB	= 2011
"SYNTAX_ENDOFSUB_RECIEVED_ENDOFTASK_EXPECTED"		
	SYNTAX_ENDOFSUB_RECIEVED_ENDOFTASK_EXPECTED	= 2012
"SYNTAX_ENDOFTASK_RECIEVED_ENDOFSUB_EXPECTED"		
	SYNTAX_ENDOFTASK_RECIEVED_ENDOFSUB_EXPECTED	= 2013
"SYNTAX_GOSUB_NOT_ALLOWED_IN_SUBS"		
	SYNTAX_GOSUB_NOT_ALLOWED_IN_SUBS	= 2014
"DOWNLOAD_ERROR"		
	DOWNLOAD_ERROR	= 2015
"DOWNLOADFIRMWARE_ERROR"		
	DOWNLOADFIRMWARE_ERROR	= 2016
"DOWNLOAD_FROM_FILE"		
	DOWNLOAD_FROM_FILE	= 2017
"DOWNLOAD_NOT_ENOUGH_MEMORY"		
	DOWNLOAD_NOT_ENOUGH_MEMORY	= 2018
"DOWNLOAD_ERROR_IN_DOWNLOAD_CHECKSUM"		
	DOWNLOAD_ERROR_IN_DOWNLOAD_CHECKSUM	= 2019
"DOWNLOAD_ERROR_IN_DOWNLOAD_RAMCHECKSUMERROR"		
	DOWNLOAD_ERROR_IN_DOWNLOAD_RAMCHECKSUMERROR	= 2020

Main thread:

"RETURN_ERROR_FROM_BRICK"
RETURN_ERROR_FROM_BRICK = 2021
"RANGE_CHECK_ERROR"
RANGE_CHECK_ERROR = 2022
“SEMANTIC_IF_ARGUMENTS_OUT_OF_RANGE"
SEMANTIC_IF_ARGUMENTS_OUT_OF_RANGE = 2023
“SEMANTIC_WHILE_ARGUMENTS_OUT_OF_RANGE"
SEMANTIC_WHILE_ARGUMENTS_OUT_OF_RANGE = 2024
"SEMANTIC_LOOP_ARGUMENTS_OUT_OF_RANGE"
SEMANTIC_LOOP_ARGUMENTS_OUT_OF_RANGE = 2025

Extra:

"DOWNLOAD_ERROR_UNKNOWN"
DOWNLOAD_ERROR_UNKNOWN = 2026
"DOWNLOAD_ALREADY_IN_DL_WHEN_RECIEVING_BEGIN"
DOWNLOAD_ALREADY_IN_DL_WHEN_RECIEVING_BEGIN = 2027
“DOWNLOAD_BRICK_IS_NOT_IN_DL_MODE"
DOWNLOAD_BRICK_IS_NOT_IN_DL_MODE = 2028
“DOWNLOAD_SYNTAX_ERROR_IN_BLOCK"
DOWNLOAD_SYNTAX_ERROR_IN_BLOCK = 2029

Appendix C – RCXdata.bas:

```
'==============================================================================
'
' Project:  MindStorms
' Unit   :  Global module
' Rev.   :  1.0
'
'------------------------------------------------------------------------------
'
' Declaration of global names for RCX-related constants
'
'==============================================================================

Option Explicit

'==============================================================================
' System sounds
'==============================================================================

Public Const CLICK_SOUND = 0
Public Const BEEP_SOUND = 1
Public Const SWEEP_DOWN_SOUND = 2
Public Const SWEEP_UP_SOUND = 3
Public Const ERROR_SOUND = 4
Public Const SWEEP_FAST_SOUND = 5

'==============================================================================
' Source names
'==============================================================================

Public Const VAR = 0
Public Const TIMER = 1
Public Const CON = 2
Public Const MOTSTA = 3
Public Const RAN = 4
Public Const TACC = 5
Public Const TACS = 6
Public Const MOTCUR = 7
Public Const KEYS = 8
Public Const SENVAL = 9
Public Const SENTYPE = 10
Public Const SENMODE = 11
Public Const SENRAW = 12
Public Const BOOL = 13
Public Const WATCH = 14
Public Const PBMESS = 15

'==============================================================================
' Sensor names
'==============================================================================

Public Const SENSOR_1 = 0
Public Const SENSOR_2 = 1
Public Const SENSOR_3 = 2

'==============================================================================
' Timer names
'==============================================================================

Public Const TIMER_1 = 0
Public Const TIMER_2 = 1
Public Const TIMER_3 = 2
Public Const TIMER_4 = 3
```

```
'================================================================================
' Tacho names (CyberMaster only)
'================================================================================

Public Const LEFT_TACHO = 0
Public Const RIGHT_TACHO = 1

'================================================================================
' Sensor types
'================================================================================

Public Const NO_TYPE = 0
Public Const SWITCH_TYPE = 1
Public Const TEMP_TYPE = 2
Public Const LIGHT_TYPE = 3
Public Const ANGLE_TYPE = 4

'================================================================================
' Sensor modes
'================================================================================

Public Const RAW_MODE = 0
Public Const BOOL_MODE = 1
Public Const TRANS_COUNT_MODE = 2
Public Const PERIOD_COUNT_MODE = 3
Public Const PERCENT_MODE = 4
Public Const CELSIUS_MODE = 5
Public Const FAHRENHEIT_MODE = 6
Public Const ANGLE_MODE = 7

'================================================================================
' Output names
'================================================================================

Public Const OUTPUT_A = 0
Public Const OUTPUT_B = 1
Public Const OUTPUT_C = 2

'================================================================================
' Logical comparison operators
'================================================================================

Public Const GT = 0
Public Const LT = 1
Public Const EQ = 2
Public Const NE = 3

'================================================================================
' Time constants
'================================================================================

Public Const MS_10 = 1
Public Const MS_20 = (2 * MS_10)
Public Const MS_30 = (3 * MS_10)
Public Const MS_40 = (4 * MS_10)
Public Const MS_50 = (5 * MS_10)
Public Const MS_60 = (6 * MS_10)
Public Const MS_70 = (7 * MS_10)
Public Const MS_80 = (8 * MS_10)
Public Const MS_90 = (9 * MS_10)
Public Const MS_100 = (10 * MS_10)
Public Const MS_200 = (20 * MS_10)
Public Const MS_300 = (30 * MS_10)
Public Const MS_400 = (40 * MS_10)
Public Const MS_500 = (50 * MS_10)
Public Const MS_700 = (70 * MS_10)
Public Const SEC_1 = (100 * MS_10)
Public Const SEC_2 = (2 * SEC_1)
Public Const SEC_3 = (3 * SEC_1)
Public Const SEC_5 = (5 * SEC_1)
Public Const SEC_10 = (10 * SEC_1)
Public Const SEC_15 = (15 * SEC_1)
Public Const SEC_20 = (20 * SEC_1)
Public Const SEC_30 = (30 * SEC_1)
Public Const MIN_1 = (60 * SEC_1)
```

Appendix D - GetStarted.bas:

```
'====================================================================================
'
' Project:  MindStorms explanatory demo project
' Unit   :  Global module
' Rev.   :  1.0
'
'------------------------------------------------------------------------------------
'
' Declaration of global names for sensors, tasks, subroutines, timers, varia-
' bels and constants.
'
'                         !!! IMPORTANT NOTICE - WARNING !!!
'
' It is the responsibility of the application programmer to allocate the program,
' sub, task and variable numbers without overlap in the individual programs.
'
'====================================================================================

Option Explicit

'====================================================================================
' Program names
'
'   Syntax: <descriptive name>Prog
'   Range: 0 to 4 (RCX), 0 (CyberMaster)
'====================================================================================

Public Const MotorControlProg = 0

'====================================================================================
' Task names
'
'   Syntax: <descriptive name>Task
'   Range: 0 to 9 (RCX), 0 to 3 (CyberMaster)
'====================================================================================

Public Const MotorOnOffTask = 0

'====================================================================================
' Subroutine names
'
'   Syntax: <descriptive name>Sub
'   Range: 0 to 7 (RCX), 0 to 3 (CyberMaster)
'====================================================================================

Public Const NiceAndHandySub = 0
Public Const UsefulAndSmallSub = 1 ' and so on

'====================================================================================
' Sensor names
'
' Syntax: s<descriptive name>
'====================================================================================

Public Const sStopButton = SENSOR_1
Public Const sSearchLight = SENSOR_2

'====================================================================================
' Output/motor names
'
' Syntax: o<descriptive name> || m<descriptive name>
'====================================================================================

Public Const oBlink = OUTPUT_A
Public Const mForward = OUTPUT_B
```

```
'==========================================================================
' Timer names
'
' Syntax: t<descriptive name>
'==========================================================================

Public Const tDrum = TIMER_1
Public Const tStick = TIMER_2 ' and so on

'==========================================================================
' Variable names
'
' Syntax: v<descriptive name>
'==========================================================================

Public Const vLeftThreshold = 0
Public Const vRightThreshold = 1

'==========================================================================
' Constant declarations
'
' Syntax: k<descriptive name>
'==========================================================================

Public Const kForever = 0
Public Const kThresholdOffset = 50

'--------------------------------------------------------------------------
' Logical constants
'--------------------------------------------------------------------------

Public Const kFalse = 0
Public Const kTrue = 1

'--------------------------------------------------------------------------
' Motor control constants
'--------------------------------------------------------------------------

Public Const kOff = 0
Public Const kOn = 1
Public Const kFullSpeed = 7

'--------------------------------------------------------------------------
' Your own stuff
'--------------------------------------------------------------------------

Public Const kFirm = 0
Public Const kFixed = 1
' and so on
```

```
'==========================================================================
' Timer names
'
' Syntax: t<descriptive name>
'==========================================================================

Public Const tDrum = TIMER_1
Public Const tStick = TIMER_2 ' and so on

'==========================================================================
' Variable names
'
' Syntax: v<descriptive name>
'==========================================================================

Public Const vLeftThreshold = 0
Public Const vRightThreshold = 1

'==========================================================================
' Constant declarations
'
' Syntax: k<descriptive name>
'==========================================================================

Public Const kForever = 0
Public Const kThresholdOffset = 50

'--------------------------------------------------------------------------
' Logical constants
'--------------------------------------------------------------------------

Public Const kFalse = 0
Public Const kTrue = 1

'--------------------------------------------------------------------------
' Motor control constants
'--------------------------------------------------------------------------

Public Const kOff = 0
Public Const kOn = 1
Public Const kFullSpeed = 7

'--------------------------------------------------------------------------
' Your own stuff
'--------------------------------------------------------------------------

Public Const kFirm = 0
Public Const kFixed = 1
' and so on
```