EV3 Basic Manual

*This manual is also available in **French**, **Spanish**, **German** and **Russian**.*

The runtime-library for the EV3 is organized in individual parts, called 'objects'. Each object provides functionality for a different part of the system. This list contains everything that is available for both Small Basic on the PC (with EV3-extension) and also on the brick when compiling with EV3 Explorer. When developing programs that only need to run on the PC, you can also use everything else that is provided by Small Basic, but that is not documented here. The objects documented here are:

Assert

A test facility to help check part of the code for correctness.
Assertions make implicit assumptions about the current program state explicit. By adding assertion calls you can help finding bugs in your program more easily. For example, when a part of the program depends on the variable A having a positive value, you could call Assert.Greater(A,0,"A must be > 0!"). In the case that the program runs into an assertion that is not satisfied, the error message is displayed stating the problem.

Assert.Equal (a, b, message)

Make sure that two values are equal. For this test, even "True" and "tRue" are not considered equal.

a First value
b Second value
message Message that will be displayed if the assertion fails.

Assert.Failed (message)

Write a failure message to the display. This function should only be called if something has already failed in the program.

message Message to be displayed

Assert.Greater (a, b, message)

Make sure that the first number is greater than the second.

a First value
b Second value
message Message that will be displayed if the assertion fails.

Assert.GreaterEqual (a, b, message)

Make sure that the first number is greater than or equal to the second.

a First value
b Second value
message Message that will be displayed if the assertion fails.

Assert.Less (a, b, message)

Make sure that the first number is less than the second.

a First value

b Second value
message Message that will be displayed if the assertion fails.

Assert.LessEqual (a, b, message)
Make sure that the first number is less than or equal to the second.

a First value
b Second value
message Message that will be displayed if the assertion fails.

Assert.Near (a, b, message)
Make sure that the two numbers are nearly identical. This can be used for fractional numbers with many decimal places where the computation could give slightly different results because of rounding issues.

a First value
b Second value
message Message that will be displayed if the assertion fails.

Assert.NotEqual (a, b, message)
Make sure that two values are not equal. For this test, even "True" and "tRue" are not considered equal.

a First value
b Second value
message Message that will be displayed if the assertion fails.

Buttons

Reads the states and clicks of the buttons on the brick. The buttons are specified with the following letters: **U**(p), **D**(own), **L**(eft), **R**(ight), **E**(nter)

Buttons.Current - property
The buttons that are currently pressed. This property contains a text with the key letters of all keys being pressed at the moment.

Buttons.Flush ()
Remove any clicked-state of all buttons. Subsequent calls to GetClicks will only deliver the buttons that were clicked after the flush.

Buttons.GetClicks ()
Checks which buttons were clicked since the last call to GetClicks and returns a text containing their letters. The 'clicked' state of the buttons is then removed. Also a sound is emitted from the brick when a click was detected.

Returns A text containing the letters of the clicked buttons (can be empty)

Buttons.Wait ()
Wait until at least one button is clicked. If a buttons was already clicked before calling this function, it returns immediately.

EV3

Small utility functions that concern the EV3 brick as a whole.

EV3.BatteryLevel - property
The current loading level of the battery (range 0 to 100).

EV3.Time - property
The time in milliseconds since the program was started.

EV3.SetLEDColor (color, effect)
Set the color of the brick LED light and the effect to use for it.

color Can be "OFF", "GREEN", "RED", "ORANGE"
effect Can be "NORMAL", "FLASH", "PULSE"

EV3.SystemCall (commandline)
Execute one system command by the command shell of the EV3 Linux system. All threads of the virtual machine are halted until the system command is finished.

commandline The system command.
Returns Exit status of the command.

EV3File

Access the file system on the EV3 brick to read or write data.
File names can be given either absolute (with a leading '/') to reach any file in the system, or relative to the 'prjs' folder.

EV3File.Close (handle)
Close an open file.

handle The file handle (previously obtained from an Open... call)

EV3File.ConvertToNumber (text)
Utility function to convert a text to a number.

text A text holding a number in decimal representation (with optional fractional digits)
Returns The number

EV3File.OpenAppend (filename)
Open a file for adding data. When the file does not exist, it will be created.

filename Name of the file to create/extend.
Returns A number that identifies this open file (a.k.a. file handle).

EV3File.OpenRead (filename)
Open a file for reading data. When the file does not exist, a 0 is returned.

filename Name of the file to read from.
Returns A number that identifies this open file (a.k.a. file handle) or 0 if file does not exist.

EV3File.OpenWrite (filename)
Open a file for writing. When the file already exists, it will be overwritten.

filename Name of the file to create/overwrite.
Returns A number that identifies this open file (a.k.a. file handle).

EV3File.ReadByte (handle)
Read one byte of data from the file.

handle The file handle (previously obtained from an Open... call)
Returns The next byte from the file.

EV3File.ReadLine (handle)
Read one line of text from the file. The line will be decoded using the ISO-8859-1 encoding and must be terminated with a newline-character (code 10).

handle The file handle (previously obtained from an Open... call)
Returns The text from the current line in the file.

EV3File.TableLookup (filename, bytes_per_row, row, column)
Utility function to read bytes from potentially huge data files that are too big to be transferred to memory as a whole. Because the file could be so big that the numerical precision of the normal numbers is not enough, a row/column addressing is possible.

filename The name of the file.
bytes_per_row When the file has a row/column structure, this is the number of bytes in one row. Use 1 if not applicable.
row Which row to access (start with 0).
column Which column to access (start with 0).
Returns The byte on the denoted position

EV3File.WriteByte (handle, data)
Write a single byte of data to the file.

handle The file handle (previously obtained from an Open... call)
data One byte to write (value of 0 - 255).

EV3File.WriteLine (handle, text)
Write one line of text to the file. The line will be encoded with ISO-8859-1 encoding and will be terminated with a newline-character (code 10).

handle The file handle (previously obtained from an Open... call)
text The text to write to the file.

LCD

Control the LCD display on the brick.
The EV3 has a black-and-white display with 178 x 128 pixels. All pixels are addressed with X,Y coordinates where X=0 is the left edge and Y=0 is the top edge. Thus the coordinates of the bottom right pixel are (177,127).

LCD.BmpFile (color, x, y, filename)
Draw a bitmap file in a given color to the display.

color 0 (white) or 1 (black)
x X coordinate of left edge
y Y coordinate of top edge
filename Name of the file containing the bitmap

LCD.Circle (color, x, y, radius)
Draws a circle in the given color.

color 0 (white) or 1 (black)
x X coordinate of center point
y Y coordinate of center point
radius Radius of the circle

LCD.Clear ()
Set all pixels of the display to white.

LCD.FillCircle (color, x, y, radius)
Draws a filled circle with a given color.

color 0 (white) or 1 (black)
x X coordinate of center point
y Y coordinate of center point
radius Radius of the circle

LCD.FillRect (color, x, y, width, height)
Fill a rectangle with a color.

color 0 (white) or 1 (black)
x Left edge of rectangle
y Top edge of rectangle
width Width of rectangle
height Height of rectangle

LCD.InverseRect (x, y, width, height)
Invert the colors of all pixels inside of a rectangle

x Left edge of rectangle
y Top edge of rectangle
width Width of rectangle
height Height of rectangle

LCD.Line (color, x1, y1, x2, y2)
Set a straight line of pixels to a color.

color 0 (white) or 1 (black)
x1 X coordinate of start point
y1 Y coordinate of start point
x2 X coordinate of end point
y2 Y coordinate of end point

LCD.Pixel (color, x, y)
Set a single pixel on the display to a color.

color 0 (white) or 1 (black)
x X coordinate
y Y coordinate

LCD.Rect (color, x, y, width, height)
Draw an outline of a rectangle with a color.

color 0 (white) or 1 (black)
x Left edge of rectangle
y Top edge of rectangle
width Width of rectangle
height Height of rectangle

LCD.StopUpdate ()
Memorize all subsequent changes to the display instead of directly drawing them. At the next call to Update(), these changes will be finally drawn. You can use this feature to prevent flickering or to speed up drawing complex things to the LCD.

LCD.Text (color, x, y, font, text)
Write a given text (or number) in a color to the display

color 0 (white) or 1 (black)
x X coordinate where text starts
y Y coordinate of the top corner
font Size of the letters: 0 (TINY), 1 (SMALL), 2 (BIG)
text The text (or number) to write to the display

LCD.Update ()
Draw all changes to the display that have happened since the last call to StopUpdate(). After Update() everything will again be drawn directly unless you use the StopUpdate() once more.

LCD.Write (x, y, text)
Write a given text (or number) in black color to the display. When you need more control over the visual appearance, use the function 'Text' instead.

x X coordinate where text starts
y Y coordinate of the top corner
text The text (or number) to write to the display

Math

The Math class provides lots of useful mathematics related methods

Math.Pi - property
Gets the value of Pi

Math.Abs (number)
Gets the absolute value of the given number. For example, -32.233 will return 32.233.

number The number to get the absolute value for.
Returns The absolute value of the given number.

Math.ArcCos (cosValue)
Gets the angle in radians, given the cosine value.

cosValue The cosine value whose angle is needed.
Returns The angle (in radians) for the given cosine Value.

Math.ArcSin (sinValue)
Gets the angle in radians, given the sin value.

sinValue The sine value whose angle is needed.
Returns The angle (in radians) for the given sine Value.

Math.ArcTan (tanValue)
Gets the angle in radians, given the tangent value.

tanValue The tangent value whose angle is needed.
Returns The angle (in radians) for the given tangent Value.

Math.Ceiling (number)
Gets an integer that is greater than or equal to the specified decimal number. For example, 32.233 will return 33.

number The number whose ceiling is required.
Returns The ceiling value of the given number.

Math.Cos (angle)
Gets the cosine of the given angle in radians.

angle The angle whose cosine is needed (in radians).
Returns The cosine of the given angle.

Math.Floor (number)
Gets an integer that is less than or equal to the specified decimal number. For example, 32.233 will return 32.

number The number whose floor value is required.
Returns The floor value of the given number.

Math.GetDegrees (angle)
Converts a given angle in radians to degrees.

angle The angle in radians.
Returns The converted angle in degrees.

Math.GetRadians (angle)
Converts a given angle in degrees to radians.

angle The angle in degrees.
Returns The converted angle in radians.

Math.GetRandomNumber (maxNumber)
Gets a random number between 1 and the specified maxNumber (inclusive).

maxNumber The maximum number for the requested random value.
Returns A Random number that is less than or equal to the specified max.

Math.Log (number)
Gets the logarithm (base 10) value of the given number.

number The number whose logarithm value is required
Returns The log value of the given number

Math.Max (number1, number2)
Compares two numbers and returns the greater of the two.

number1 The first of the two numbers to compare.
number2 The second of the two numbers to compare.
Returns The greater value of the two numbers.

Math.Min (number1, number2)
Compares two numbers and returns the smaller of the two.

number1 The first of the two numbers to compare.
number2 The second of the two numbers to compare.
Returns The smaller value of the two numbers.

Math.NaturalLog (number)
Gets the natural logarithm value of the given number.

number The number whose natural logarithm value is required.
Returns The natural log value of the given number.

Math.Power (baseNumber, exponent)
Raises the base number to the specified power.

baseNumber The number to be raised to the exponent power.
exponent The power to raise the base number.
Returns The base number raised to the specified exponent.

Math.Remainder (dividend, divisor)
Divides the first number by the second and returns the remainder.

dividend The number to divide.
divisor The number that divides.
Returns The remainder after the division.

Math.Round (number)
Rounds a given number to the nearest integer. For example 32.233 will be rounded to 32.0 while 32.566 will be rounded to 33.

number The number whose approximation is required.
Returns The rounded value of the given number.

Math.Sin (angle)
Gets the sine of the given angle in radians.

angle The angle whose sine is needed (in radians)
Returns The sine of the given angle

Math.SquareRoot (number)
Gets the square root of a given number.

number The number whose square root value is needed.
Returns The square root value of the given number.

Math.Tan (angle)
Gets the tangent of the given angle in radians.

angle The angle whose tangent is needed (in radians).
Returns The tangent of the given angle.

Motor

Note that EV3 Basic makes no distinction between large and medium motors.

The Motor commands control the motors connected to the brick. For every command you need to specify one or more motor ports that should be affected (for example, "A", "BC", "ABD"). When additional bricks are daisy-chained to the master brick, address the correct port by adding the layer number to the specifier (for example, "3BC", "2A"). In this case only the motors of one brick can be accessed with a single command.

Speed vs. Power: When requesting to drive a motor with a certain **speed**, the electrical power will be permanently adjusted to keep the motor on this speed regardless of the necessary driving force (as long as enough power can be provided). When requesting a certain **power** instead, the motor will just be provided with this much electrical power and the actual speed will then depend on the resistance it meets. The normal choice in EV3 Basic should be **speed**.

The following table summarises the nine commands that can be used to make motors move:

	Move x degrees (program waits for completion)	**Start to run indefinitely**	**Start to move x degrees**
Regulate speed	Motor.Move	Motor.Start	Motor.Schedule
Choose power	Motor.MovePower	Motor.StartPower	Motor.SchedulePower
Synchronize	Motor.MoveSync	Motor.StartSync	Motor.ScheduleSync

The four highlighted commands are the most basic ones, the ones beginners will use most of the time.

To make one motor turn through a given angle or make two motors turn at the same speed through a given angle (as for straight line motion), prefer **Motor.Move**.

To make the robot follow a curved path for a given angle of wheel rotation (two motors moving with different speeds), prefer **Motor.MoveSync**. This function causes program execution to pause until the movement completes. This function is equivalent to the 'move tank' block of the standard Lego software.

To make motors turn for a given length of time, turn them on with **Motor.Start** (to give motors the same speed) or **Motor.StartSync** (to give motors different speeds), then use **Program.Delay** to make the program wait the desired time, then stop the motors with **Motor.Stop**.

When you have to change power or speed of an already running motor, just re-issue a Start() command with the appropriate speed or power value. The motor will then seamlessly switch over to the new mode of operation.

For **all** the motor commands:

- **port(s)** = Motor port name(s). E.g. "BC"
- **degrees** = Number of degrees to move the motor. Motor commands will always use the *absolute value* of the degree parameter (in other words any negative sign will be ignored), so **if you want the motor to turn backwards give it a negative power or speed rather than a negative angle**.
- **brake** = "True", if the motor(s) should switch on the brake after movement
- **speed** = speed from -100 (full reverse) to 100 (full forward)
- **power** = Power level from -100 (full reverse) to 100 (full forward)

Motor.GetCount (port)
Query the current rotation count of a single motor. As long as the counter is not reset it will accurately measure all movements of a motor, even if the motor is driven by some external force while not actively running.

Returns The current rotation count in degrees.

Motor.GetSpeed (port)
Query the current speed of a single motor.

Returns Current speed in range -100 to 100

Motor.IsBusy (ports)
Checks if one or more motors are still busy with a scheduled motor movement.

Returns "True" if at least one of the motors is busy, "False" otherwise.

Motor.Move (ports, speed, degrees, brake)
Move one or more motors with the specified *speed* through the specified angle (in degrees). This command will cause the program to pause until the motor has reached its destination. When you need finer control over the movement (soft acceleration or deceleration), consider using the command Motor.SchedulePower instead. For the difference between 'speed' and 'power', see the beginning of this Motor section.

speed Speed level from -100 (full reverse) to 100 (full forward)
degrees The angle to rotate. Any negative sign will be ignored.

Motor.MovePower (ports, power, degrees, brake)
Move one or more motors with the specified *power* the specified angle (in degrees). This command will block execution until the motor has reached its destination. When you need finer control over the movement (soft acceleration or deceleration), consider using the command Motor.SchedulePower() instead.

power Power level from -100 (full reverse) to 100 (full forward)
degrees The angle to rotate. *Any negative sign will be ignored.*

Motor.MoveSync (ports, speed1, speed2, degrees, brake)
This function is similar to the 'Move Tank' block in the standard Lego EV3 software. Moves 2 motors (with speeds that can be different) a defined number of degrees. The angle to move will be measured at the motor with the higher speed. This function pauses program execution until the movement completes. If you want the movement to return immediately you should use Motor.ScheduleSync() instead. The two motors are *synchronized* which means that when one motor experiences some resistance and can not keep up its speed, the other motor will also slow down or stop altogether. This is especially useful for vehicles with two independently driven wheels which still need to go straight or make a specified turn.

ports Names of exactly TWO motor ports (for example "AB" or "CD")

speed1 Speed value from -100 (full reverse) to 100 (full forward) of the motor with the *lower* port letter.
speed2 Speed value from -100 (full reverse) to 100 (full forward) of the motor with the *higher* port letter.
degrees The angle that the *faster* motor should rotate. *Any negative sign will be ignored*.

Motor.ResetCount (ports)
Set the rotation count of one or more motors to 0.

Motor.Schedule (ports, speed, degrees1, degrees2, degrees3, brake)
Move one or more motors with the specified speed values. The speed can be adjusted along the total rotation to get a soft start and a soft stop if needed. The angle to rotate the motor is degrees1+degrees2+degrees3. Any negative signs for the angles will be ignored. At the end of the movement, the motor stops automatically (with or without using the brake). This function returns immediately, so if you want the movement to complete before the program continues you should follow this command with Motor.Wait(ports).

speed Speed level from -100 (full reverse) to 100 (full forward)
degrees1 The part of the rotation with acceleration
degrees2 The part of the rotation with uniform speed
degrees3 The part of the rotation with deceleration

Motor.SchedulePower (ports, power, degrees1, degrees2, degrees3, brake)
Move one or more motors with the specified power. The power can be adjusted along the total rotation to get a soft start and a soft stop if needed. The angle to rotate the motor is degrees1+degrees2+degrees3. Any negative signs for the angles will be ignored. At the end of the movement, the motor stops automatically (with or without using the brake). This function returns immediately, so if you want the movement to complete before the program continues you should follow this function with **Motor.Wait(ports)**.

power Power level from -100 (full reverse) to 100 (full forward)
degrees1 The part of the rotation with acceleration
degrees2 The part of the rotation with uniform motion
degrees3 The part of the rotation with deceleration

Motor.ScheduleSync (ports, speed1, speed2, degrees, brake)
Move 2 motors with speeds that can be different through a defined number of degrees. The 'degrees' value is applied to the motor with the higher speed. The two motors are *synchronized*, which means that when one motor experiences some resistance and can not keep up its speed, the other motor will also slow down or stop altogether. This is especially useful for vehicles with two independently driven wheels which still need to go straight or make a specified turn. This function returns immediately. If you want the movement to complete before the program continues you should use **Motor.MoveSync()** instead.

ports Names of exactly 2 motor ports (for example "AB" or "CD")
speed1 Speed value from -100 (full reverse) to 100 (full forward) of the motor with the lower port letter.
speed2 Speed value from -100 (full reverse) to 100 (full forward) of the motor with the higher port letter.
degrees The angle of the faster motor to rotate. Any negative sign will be ignored.

Motor.Start (ports, speed)
Start one or more motors with the requested speed or set an already running motor to this speed.

speed Speed value from -100 (full reverse) to 100 (full forward).

Motor.StartPower (ports, power)
Start one or more motors with the requested power or set an already running motor to this power.

power Power value from -100 (full reverse) to 100 (full forward).

Motor.StartSync (ports, speed1, speed2)
Set two motors to run synchronized at their chosen speed levels. The two motors will be synchronized, that means, when one motor experiences some resistance and can not keep up its speed, the other motor will also slow down or stop altogether. This is especially useful for vehicles with two independently driven wheels which still need to go straight or make a specified turn. The motors will keep running until stopped by another command.

ports Name of exactly two motor ports (for example "AB" or "CD").
speed1 Speed value from -100 (full reverse) to 100 (full forward) of the motor with the lower port letter.
speed2 Speed value from -100 (full reverse) to 100 (full forward) of the motor with the higher port letter.

Motor.Stop (ports, brake)
Stop one or multiple motors. This will also cancel any scheduled movement for this motor.

Motor.Wait (ports)
Wait until the specified motor(s) has finished a "Schedule..." or "Move..." operation. Using this method is normally better than calling IsBusy() in a tight loop.

Program

The Program class provides helpers to control the program execution.

Program.ArgumentCount - property
Gets the number of command-line arguments passed to this program.

Program.Directory - property
Gets the executing program's directory.

Program.Delay (milliSeconds)
Delays program execution by the specified number of milliseconds.

milliSeconds The amount of delay.

Program.End ()
Ends the program.

Program.GetArgument (index)
Returns the specified argument passed to this program.

index Index of the argument.
Returns The command-line argument at the specified index.

Sensor

Access sensors that are attached to the brick. To specify the sensor use the port number which is printed below the socket on the brick (for example 1). To access sensors of further bricks that are connected via daisy-chaining, use the next higher numbers instead (5 - 8 will access the sensors on the first daisy-chained brick, 9-12 the sensors on the next one and so on).

Sensor.CommunicateI2C (port, address, writebytes, readbytes, writedata)

Microsoft's Ed Price, writing about EV3 Basic, explains I2C like this: *I2C (Inter-Integrated Circuit), pronounced I-squared-C, is a multi-master, multi-slave, single-ended, serial computer bus. It is typically used for attaching lower-speed peripheral ICs to processors and microcontrollers. It is a 2 line (plus common ground) communication method for one master device to control up to 112 slave devices.*

The Sensor.CommunicateI2C function communicates with devices using the I2C protocol over one of the sensor ports. This command addresses one device on the I2C-bus and can send and receive multiple bytes. This feature could be used to attach a custom sensor or to communicate with any device that is capable to be connected to the I2C bus as a slave.

port Number of the sensor port
address Address (0 - 127) of the I2C slave on the I2C bus
writebytes Number of bytes to write to the slave (maximum 31).
readbytes Number of bytes to request from the slave (maximum 32, minimum 1).
writedata Array holding the data bytes to be sent (starting at 0).
Returns An array holding the requested number of values. Index starts at 0.

Sensor.GetMode (port)

Get current operation mode of a sensor. Many sensors can work in substantially different modes. For example, the color sensor can detect ambient light, reflected light or color). When the sensor is plugged in it will normally start with mode 0, but that can be later changed by the program.

port Number of the sensor port
Returns Current operation mode (0 is always the default mode)

Sensor.GetName (port)

Get the name and mode of a currently connected sensor. This function is mainly intended for diagnostic use because you normally know which sensor is plugged to which port on the model.

port Number of the sensor port
Returns Description text (for example, "TOUCH")

Sensor.GetType (port)

Get the numerical type identifier of a currently connected sensor.

port Number of the sensor port
Returns Sensor type identifier (for example, 16 for a touch sensor)

Sensor.IsBusy (port)

Check if a sensor is currently busy switching mode or in process of initialization. After mode switching a sensor may take some time to become ready again.

port Number of the sensor port
Returns "True" if the sensor is currently busy

Sensor.ReadPercent (port)

Read the current sensor value and apply some sensible percentage scaling. Most sensors can translate the current reading to a meaningful single percentage value like light intensity or button press state.

port Number of the sensor port
Returns The percentage value (for example, the touch sensor gives 100 for pressed and 0 for non pressed)

Sensor.ReadRaw (port, values)

Read current sensor value where the result from ReadPercent() is not specific enough. Some sensor modes deliver values that cannot be translated to percentage

(for example a color index) or that contain multiple values at once (for example the individual red, green, blue light intensities).

port Number of the sensor port
values Requested size of result array
Returns An array holding the requested number of values. Index starts at 0. Elements that received no data are set to 0.

Sensor.ReadRawValue (port, index)

Similar to ReadRaw, but returns only a single raw value instead of an array of raw values.

port Number of the sensor port
index Index of the value that should be picked from the result array (starting with index 0). **The index value should always be 0** unless you are using the infrared sensor in remote control mode - see the **sensor appendix**.
Returns One element of a raw sensor reading.

Sensor.SetMode (port, mode)

Switches the mode of a sensor. Many sensors can work in different modes giving quite different readings. The meaning of each mode number depends on the specific sensor type. For further info, see the sensor list in the **appendix**.

port Number of the sensor port
mode New mode to switch to. This only succeeds when the mode is indeed supported by the sensor.

Sensor.Wait (port)

Wait until a sensor has finished its reconfiguration. When no sensor is plugged into the port, this function returns immediately.

port Number of the sensor port

Speaker

Use the built-in speaker of the brick to play tones or sound files.

Speaker.IsBusy ()

Check if the speaker is still busy playing a previous sound.

Returns "True", if there is a sound still playing, "False" otherwise.

Speaker.Note (volume, note, duration)

Start playing a simple tone defined by its text representation.

volume Volume can be 0 - 100
note Text defining a note "C4" to "B7" or half-tones like "C#5"
duration Duration of the tone in milliseconds

Speaker.Play (volume, filename)

Start playing a sound from a sound file stored on the brick. You need a special sound format called RSF which can only be created by the Lego software.

volume Volume can be 0 - 100
filename Name of the sound file *without the .rsf extension*

Speaker.Stop ()

Stop any currently playing sound or tone.

Speaker.Tone (volume, frequency, duration)

Start playing a simple tone of defined frequency.

volume Volume can be 0 - 100
frequency Frequency in Hz can be 250 - 10000
duration Duration of the tone in milliseconds

Speaker.Wait ()
Wait until the current sound has finished playing. When no sound is playing, this function returns immediately.

Text

The Text object provides helpful operations for working with Text.

Text.Append (text1, text2)
Appends two text inputs and returns the result as another text. This operation is particularly useful when dealing with unknown text in variables which could accidentally be treated as numbers and get added, instead of getting appended.

text1 First part of the text to be appended.
text2 Second part of the text to be appended.
Returns The appended text containing both the specified parts.

Text.ConvertToLowerCase (text)
Converts the given text to lower case.

text The text to convert to lower case.
Returns The lower case version of the given text.

Text.ConvertToUpperCase (text)
Converts the given text to upper case.

text The text to convert to upper case.
Returns The upper case version of the given text.

Text.EndsWith (text, subText)
Gets whether or not a given text ends with the specified subText.

text The larger text to search within.
subText The sub-text to search for.
Returns True if the subtext was found at the end of the given text.

Text.GetCharacter (characterCode)
Given the Unicode character code, gets the corresponding character, which can then be used with regular text.

characterCode The character code (Unicode based) for the required character.
Returns A Unicode character that corresponds to the code specified.

Text.GetCharacterCode (character)
Given a Unicode character, gets the corresponding character code.

character The character whose code is requested.
Returns A Unicode based code that corresponds to the character specified.

Text.GetIndexOf (text, subText)
Finds the position where a sub-text appears in the specified text.

text The text to search in.
subText The text to search for.
Returns The position at which the sub-text appears in the specified text. If the text doesn't appear, it returns 0.

Text.GetLength (text)
Gets the length of the given text.

text The text whose length is needed.
Returns The length of the given text.

Text.GetSubText (text, start, length)
Gets a sub-text from the given text.

text The text to derive the sub-text from.
start Specifies where to start from.
length Specifies the length of the sub text.
Returns The requested sub-text

Text.GetSubTextToEnd (text, start)
Gets a sub-text from the given text from a specified position to the end.

text The text to derive the sub-text from.
start Specifies where to start from.
Returns The requested sub-text.

Text.IsSubText (text, subText)
Gets whether or not a given subText is a subset of the larger text.

text The larger text within which the sub-text will be searched.
subText The sub-text to search for.
Returns True if the subtext was found within the given text.

Text.StartsWith (text, subText)
Gets whether or not a given text starts with the specified subText.

text The larger text to search within.
subText The sub-text to search for.
Returns True if the subtext was found at the start of the given text.

Thread

This object supports the use of threads in a program. A thread is a part of program code that can run independently and at the same time as other parts of the program. For example, you could create a thread that controls the motors, while a different thread can watch sensors or user input. Generally speaking, multi-threading is quite a complex topic. To really understand it, some extra study is recommended.

Thread.Run - property
With this property, new threads are created. Just assign a subprogram to this and the subprogram will start running as an independent thread (for example, Thread.Run = MYSUB). Any subprogram can be used to create an independent thread, but you can start the same subprogram only as one thread. A second use of Thread.Run, while the specified subprogram is still running, will just add the call to a queue that is processed after the previous run was finished. No runs will be lost in this case, but probably scheduled for a later time. Note that even in the presence of running threads, the whole program stops as soon as the main program runs to its end.

Thread.CreateMutex ()
Create a mutex (short for "mutual exclusion" handler) that can be used for thread synchronization. Only creation of mutexes is supported, but no deletion. Best practice is to create all needed mutexes at program start and keep their numbers in global variables.

Returns A number specifying the new mutex. Use this for calls to Lock and Unlock

Thread.Lock (mutex)
Tries to lock the given mutex exclusively so no other thread can aquire a lock on it. When another thread already holds a lock on the mutex, the current thread will wait until the lock is released and then acquire the lock itself (once the function call returns, the mutex has been successfully locked). This locking mechanism is normally used to protect some data structures or other resources to be accessed by two threads concurrently. Every call to Lock must be paired with a call to a subsequent Unlock.

mutex The number of the mutex, as returned from CreateMutex()

Thread.Unlock (mutex)
Releases a lock on a mutex. This function must only be called when there was indeed a preceding call to Lock.

mutex The number of the mutex, as returned from CreateMutex()

Thread.Yield ()
Explicitly gives up control of the CPU so other threads may do their work. Threads are often not really running in parallel because there may be not enough CPUs to exclusively do the work for each thread. Instead, the CPU will do a bit of work on one thread and then jump to the next thread and so on very quickly, to make it look like everything is running in parallel. Whenever a thread has nothing to do just now, but needs to wait for some condition to arrive, it can give up the control of the CPU with the Yield() function, so other threads get the chance to do their work.

Vector

This object allows direct manipulation of larger quantities of numbers.
These are called vectors and will be stored using arrays with consecutive indices (starting at 0). When arrays with different content are given to the operations, every missing array element with be treated as being 0.

Vector.Add (size, A, B)
Adds two vectors by adding the individual elements (C[0]=A[0]+B[0], C[1]=A[1]+B[1]...)

size That many numbers are taken for computation
A First vector
B Second vector
Returns A vector of the given size what contains sum values.

Vector.Init (size, value)
Set up a vector of a given size with all elements set to the same value.

size Size of the vector
value The value to use for all elements
Returns The created vector

Vector.Multiply (rows, columns, k, A, B)
Matrix multiplication operation. The input vectors are treated as two-dimensional matrices of given width and height. The individual rows of the matrix are stored inside the vectors directly one after the other. To learn more about this mathematical operation see en.wikipedia.org/wiki/Matrix_multiplication .

rows Number of rows in the resulting output matrix
columns Number of columns in the resulting output matrix
k Number of columns in input matrix A and number of rows in input matrix B
A A matrix of size rows * k
B A matrix of size k * columns
Returns A matrix holding the multiplication result

Vector.Sort (size, A)

Sort the elements of a vector in increasing order.

size Number of elements to sort
A The array containing the elements
Returns A new vector with the elements in correct order

Subpages (4): EV3 Basic handbuch EV3 Basic manuel Manual EV3 Basic Manual in Russian

Comments

You do not have permission to add comments.