Robix Scripting Reference

Feedback: desk@robix.com Home: www.Robix.com Last mod: 2005-02-14

The Robix script language is designed for programming motion sequences and setting motion parameters. The language has intentionally been kept simple.

> While many interesting and useful programs can be written directly in scripts, the script language intentionally lacks general programming constructs including variables, conditionals (e.g. "if"), loop constructs (e.g. “while”, “do”, “for”), function calls, etc. These missing pieces may be added through general-purpose programming languages such as Java, C++, Visual Basic, or any other language that can link to a C-style library.

The script language is case-insensitive, so you can use upper or lower case, or any combination, as is convenient. You may also add extra spaces between elements of a command for clarity, though a single command may not be split onto two lines.

> The notation:
>
> **<servo list>**
>
> which appear in many places below indicates that a list such as:
>
> **1,2,4**
>
> is needed. The list shown above means that the servos 1, 2 and 4 are to be used by the command. A servo list may have only one servo, or may consist of the single word:
>
> **all**
>
> which means that all servos in the pod are involved. Servos are numbered starting at 1. Similarly,
>
> **<digout list>**
>
> refers to a list of digital outputs (USB controller only) and
>
> <**aux list>**
>
> refers to a list of auxiliary outputs (LPT controller only).

The notation:

<value>

means that a number (or sometimes a keyword) is expected. The specifics of the number (or keywords) are discussed in the descriptions accompanying commands below. For example:

accel <servo list> <value>

in an actual command might be:

accel 1, 2 100

where the servo list is 1 and 2, and <value> became 100, meaning that acceleration for servos 1 and 2 is set to 100. Note that the angle brackets <> are not used in an actual command.

Multiple commands may generally appear on a single line, separated by semicolon characters '**;**'.

The units of position, speed and acceleration in the Rascal are not related to any specific unit system and so are arbitrary although they are internally consistent: a move of 50 will rotate a servo twice as far as a move of 25, within the servo's own physical limits of positional accuracy. An acceleration of 10 will bring the servo up to its maximum speed a given motion in half the time that an acceleration of 5 would take.

The complete script command set follows.

ACCEL

Syntax

accel <servo list> <value>

Description

Sets the acceleration of the servos in **<servo list>** to **<value>**. **<value>** must be in the range 1..10000.

See also: The **accdec** command for further discussion.

ACCDEC

Syntax

accdec <servo list> <value>

Description

Sets both the acceleration and deceleration of the servos in **<servo list>** to **<value>**.

<value> must be in the range 1..10000

Proper setting of acceleration and deceleration of a servo can help minimize "oscillation" of the servo as it moves. This becomes more important when the servo is moving a sizable mass.

Acceleration and deceleration may be set independently by the **accel** and **decel** commands. Making the two values different is helpful, for example, in creating a "snapping of the wrist" motion as when striking a drum with a stick.

AUXA, AUXB (used with LPT controller only)

Syntax

```
auxa on|off
auxb on|off
```

Description

Turns Auxiliary output A or B on or off.

The output is useful for powering small loads, such as LED's, postage stamp motors, etc.

DECEL

Syntax

decel <servo list> <value>

Description

Sets the deceleration of the servos in **<servo list>** to **<value>**.

<value> must be in the range 1..10000

See the **accdec** command for further discussion.

END

see **macro**

ENUMBASE (Used with DOS software only)

Syntax

`enumbase 0|1`

Description

(Note: This command is for advanced users)

Changes the numbering of servos, sensors, parameters, etc. to zero-based or one-based, according to **`<value>`** which may be either **`1`** or **`0`**.

The default value is **`1`**.

If you work in a language such as C you may prefer to number arrays starting at 0, and may wish to adopt that convention here. This is particularly true if you are also controlling the robot from C programs.

If you always want to use an **`enumbase`** of 0, you can edit the `rbx.bat` file, adding a `-e0` command line argument to the Device Driver `rbxdrv.exe`.

It is recommended that you decide on basing as early as possible, since changing the base after scripts have been created will cause those scripts to have syntax errors or operate incorrectly.

FORGET (Used with DOS software only)

Syntax

```
forget <macro name>
forget all
```

Description

The first form deletes the **<macro name>** and any other macro that was defined after it.

The **forget all** form deletes all macros.

INITPOS

Syntax

initpos <servo list> <value>

Description

Sets the servos' initial position, that is, where the servos will go when a restart command is executed.

Sets the initial position of the servos in **<servo list>** to **<value>**.

<value> may be positive or negative. Values above a servo's **maxpos** for a servo will be treated as if the value had been that servos **maxpos**. Similarly, values below **minpos** will be treated as **minpos**.

See also: **restart** command.

INVERT

Syntax

invert <servo list> on|off

Description

(Note: This command is for intermediate and advanced users.)

Turn inversion on or off for each servo in **<servo list>.**

Inverting a servo effectively makes it run counter to its normal rotation. This can be useful when a servo's uninverted (default) direction feels backwards to the programmer.

When a servo is inverted, relative move and jump commands (using the keyword by) that used to turn the servo clockwise will turn it counterclockwise, and vice versa. Similarly, moves and jumps to a specific position will result in motion to a position on the other side of p0pos, but at an equal distance from it, within the accuracy limits of the servo.

p0pos, the physical zero position, is not affected by the invert command.

At the time a servo's invert setting is changed, its **pos**, **initpos, minpos** and **maxpos** all take on the opposite sign. Their physical positions don't change.

In practice it is common to put commands like the following at the top of a script.

```
invert all off  # clear current settings
invert x,x,x on # invert servos x,x,x
```

MACRO ... ; END

Syntax

macro <name> ; <command> ; .. <command> ; end;

Description

A **macro** is a series of script commands that are executed by using the macro name as a command.

LPT software only, does *not* apply to USB software: When defining a **macro**, the **macro** command must be the first on its line. It labels the following commands, up to the end command, as a new command, <name>. The **end** command must be the last command on its line.

The commands in between **macro** and **end** may occupy multiple lines, or an entire **macro** may occupy only a single line.

A **macro** is executed by using its name as a command, optionally followed by a count indicating the number of times to execute the macro. A count of **0** indicates indefinite repetition of the **macro**, or until the <ESC> key is pressed or (for non-DOS only) a stop command is issued by mouse action. For example:

```
macro pen_up ;     move 3 to 50; end
macro pen_down; move 3 to -140; end
macro pen_tap; pen_down; pen_up; end

pen_tap          # tap the pen one time
pen_tap 1        # tap the pen one time
pen_tap 10 # tap the pen ten time
pen_tap 0        # tap the pen indefinitely

macro dot_space # multi line;
                      # indented for clarity
     pen_tap;         # make a dot
     move 4 by 10;    # space between dots
end;

dot_space 5      # make 5 dots with spaces
```

For advanced programmers:

Macros may be called from your Java (w/ USB version only), C++ or Visual Basic programs, or any other programming language that can link with a C-style library. See API's under Reference section of www.robix.com.

MAXPOS

Syntax

maxpos <servo list> <value>

Description

Sets the maximum position that will be allowed during **move** or **jump** commands for the servos in the list.

<value> can also be the keyword **default** which sets **maxpos** to 1400.

If a **move** or **jump** command would cause a servo to go past **maxpos** in a positive direction, the move command will be interpreted as a move to the maximum position.

When a **maxpos** command is executed, either **initpos** or **minpos** or both may be greater than the new value of **maxpos**, which would be inconsistent. In this case, one or both will be set equal to the new value for **maxpos**.

MAXSPD

Syntax

maxspd <servo list> <value>

Description

Sets the maximum speed that a servo may reach during execution of a move command. The **maxspd** of a servo may not actually be reached during a move command, but it will not be exceeded.

Note that during a r**estart** or **jump** command, **maxspd** may be exceeded since the servos move to their target positions **as quickly as possible.**

MINPOS

Syntax

minpos <servo list> <value>

Description

Sets the minimum position that will be allowed for the servos in the s**ervo list**.

<value> can also be the keyword **default** which sets **minpos** to -1400.

If a **move** or **jump** command would cause a servo to go past **minpos** in a negative direction, the command will be interpreted as a **move** or j**ump** to the minimum position.

When a **minpos** command is executed, either **initpos** or **minpos** or both may be below the new value of **minpos**, which would be inconsistent. In this case, one or both will be set equal to the new value of **minpos**.

MOVE

Syntax

```
move <servo list> to <value>
move <servo list> by <value>
move <slist> to|by <val>[[,<slist> to|by <val>]..]
```

Description

The first type of **move** command, called an absolute **move**, is of the form:

```
move <servo list> to <value>
```

which moves each listed servo to the position given by <value>.

The second type of **move** command, called a relative **move**, is of the form.

```
move <servo list> by <value>
```

which moves each listed servo by the amount **<value>**.

For example,

```
move 1,2,3 to 0
```

would move the listed servos to position 0,
while:

```
move 3 by -50
```

would move servo 3 by 50 units negatively.

If several servos need to move at once to different positions and/or by different amounts, then the third and most general form of the **move** command is used. Note that this form allows a combination of absolute and relative moves. In the example:

```
move 1 by 20, 2 to -400, 5,6 to -100
```

servo 1 will move by 20 units, 2 will move to position -400 and servos 5 and 6 will move by -100.

If a **move** command would carry a servo beyond its **maxpos** or **minpos**, that servo's portion of the **move** is interpreted as a **move to** the **maxpos** or **minpos**, as appropriate.

Servos included in a single **move** command move so that they start and stop together with "coordinated" motion. Each servo accelerates according to its **accel** setting and decelerates according to its **decel** setting, and for each servo its speed does not exceed (and may not reach) the servo **maxspd**.

For an absolute **move** or **jump**, **<value>** may be the word **initpos**, **maxpos** or **minpos**, causing each listed servo to move to the appropriate position. Thus, the command:

```
jump all to initpos
```

would have an effect similar to the **restart** command.

P0POS

Syntax

`p0pos <servo list> <value>`

Description

(Note: This command is for advanced users.)

Set the "physical zero position" (hence the name "p0pos") of the listed servos to the physical position given by **`<value>`**.

This command is typically used when hobby servos other than the recommended servos Hitec HS422 or similar are used. For example, if Futaba servos are used then p0pos, normally 3000, should probably be adjusted to 2400, the center position of Futaba servos in general.

PAUSE (Used with LPT Controller only)

Syntax

pause <value>

Description

Pause approximately **<value>** tenths of a second before executing the next command.

See also: **wait** command, used with USB controller only.

POWER

SAFETY NOTE:: THIS COMMAND DOES NOT TURN OFF THE SERVO POWER SUPPLY, SO IN CASE OF MALFUNCTION THE ROBOT MIGHT BEGIN TO MOVE SUDDENLY AND WITHOUT WARNING. TO COMPLETELY REMOVE POWER FROM THE ROBOT YOU MUST UNPLUG THE POWER CORD.

Syntax

`power <servo list> on|off`

Description

(Note: this command is for advanced users.)

Applies power to (**`on`**) or removes power from (**`off`**) the listed servos. When power to a servo is turned off, the servo becomes "relaxed" or "compliant".

`Move` and **`jump`** commands may be applied to servos with their **`power off`**. When **`power`** is restored, the servos will move immediately to the position implied by the series of **`move`** and j**`ump`** commands, and shown as **`pos`** in the status window, if visible. Even if the servos have been moved manually to a new position while power is off, when power is restored they will move suddenly to their **`pos`**'s.

RESTART

Syntax

`restart`

Description

Moves all servos immediately to their initial positions, as specified by the **`initpos`** value in the status window, if shown..

This command is typically run at the beginning of a robotic session to bring the robot from whatever position it currently has to its initial position.

Execution of the restart command may cause motion which is fast and sudden as the robot jumps all servos to their initial positions.

LPT Controllers only: If power to the controller is physically interrupted, the **`restart`** command should be executed either from a menu or in a script since in LPT controllers the restart command also reinitiallizes the controller.

WAIT (Used with USB Controller only)

Syntax

`wait <value>`

Description

Wait approximately **`<value>`** tenths of a second before executing the next command.

See also: p**`ause`** command, used with LPT controller only

Note: The command name was changed from 'pause' to 'wait' because of some confusion between the command and the on-screen 'pause' button.