**Brief description of the programming language**

**Introduction**

**variables:**

all variables must start and end with a dollar sign.

Example:

$x$

Variables are case sensitive, so $x$ is different from $X$

The variable $return$ is RESERVED. It is used to return values from macros.

Variables named $1$, $2$... are automatically assigned when parameters are sent to a macro.

To assign an initial value to a variable the eval command must be used:

eval $x$,1

corresponds to $x$=1

**comments:**

Text following the semicolon (;) are considered as a comment and ignored

**log file:**

When the program is executed and the connect button is pressed the variables recorded (temperature etc.) will be automatically appended to the log.txt file every 500 ms.

The following header is added first:

----------------------------------

-         PROCESS STARTS         -

----------------------------------

date time

When the disconnect button is pressed the logging is interrupted and the following text is appended to log.txt:

---------------------------------

-         PROCESS ENDED         -

---------------------------------

date time

This helps to find the log data in the file.

***Commands list***

**log**

Print string to log file

use: log text

The command adds the text to the log queue.

variables contained in text will be replaced automatically.

Example1:

log sample taken

it writes 'sample taken' into the log queue

Example2:

eval $x$,1

log x=$x$

it writes 'x=1' in the log queue

**send**

Send a Gcode command

use: send command,where

where indicates the number of the device for gcode output (0=SyringeBot, 1=Robot).

Variables are automatically substituted with their values before sending to destination.

Example:

send G28 X Y,1

Please note that due to unsynchronized connections, only a few commands (<10) can be sent in a sequence. For better performances and longer sequences use commands **buffer** and **print**

Example:

buffer

macro "syr1\_X\_ml"30

macro "syr1\_X\_ml"60

macro "syr1\_X\_ml"60

macro "purge syringe 1"

print

**buffer**

Buffer all commands; send later (with print). Used for long gcode sequences where the base command **send** will fail.

**print**

Send all buffered commands (see **buffer** command). Used for long gcode sequences where **send** will fail.

**ask**

Ask to the user the insertion of a numerical input.

use: ask $varname$,title,question,initialvalue,minvalue,maxvalue

The procedure creates a window with title *title*, outputs the text contained in *question*, sets the initial value to *initialvalue* and limits the input range from *minvalue* to *maxvalue*.

Example:

buffer

ask $ML$,Syringe 1,Enter how many ml to put,10,10,60

macro "syr1\_X\_ml"$ML$

print

**eval**

Evaluate a mathematical expression and assign the result to a variable.

use: eval $varname$,math\_expression

Example:

ask $a$,a value,Enter the value of a,1,1,10

ask $b$,b value,Enter the value of b,1,1,10

eval $sum$,$a$+$b$

echo the sum of a and b is $sum$

**exec**

Executes Python's code. Code might contain variables ($..$). Code on multiple lines must be separated with /n

use: exec code!,varname1=$var1$,...

**getsyringeparms**

Get syringe parameters for the syringe *syringenumber*.Parameters taken are: syringe 0 to max sign distance (mm), syringe total volume (ml), volume of inlet tube and volume of outlet tube as loaded in configuration.txt. The command creates and assign the following variables:

|  |  |
| --- | --- |
| **Variable Name** | **Meaning** |
| $numsyringes$ | Total number of syringes installed on Syringebot |
| $syringemax$ | Max distance in mm for syringe *syringenumber* |
| $syringevol$ | Max volume in ml for syringe *syringenumber* |
| $volinlet$ | Volume in ml of the inlet tube |
| $voloutlet$ | Volume in ml of the outlet tube |
| $axisname$ | Axis to be moved for *syringenumber* |

After this command it is safe to proceed with the calculations for the syringe movements.

use: getsyringeparms syringenumber

**macro**

Call an existing macro

use: macro "macroname" var1,var2...

This command executes a macro with the name macroname.

All the macro files are saved in the directory macros with the extensions .txt and they can be edited with a plain text editor. Macro name can contain spaces.

It is possible to pass parameters to a macro by indicating their values: var1,var2...

Example:

macro "syr1\_X\_ml"20

If parameters are passed (var1,var2...) variables $1$... will be automatically created and assigned.

In the example above, the macro syr1\_X\_ml is called and the value 20 is inserted in the variable $1$ of the macro

**echo**

Print to the shell console. Variables are automatically substituted with their values before printing.

use: echo text $varname$

Example:

ask $X$,Vial X position,Enter vial X position,1,1,8

ask $Y$,Vial Y position,Enter vial Y position,1,1,6

eval $X$,58-($X$-1)\*15.71328

eval $Y$,121+($Y$-1)\*14.8

echo X=$X$ Y=$Y$

**message**

Like **echo** but outputs the message as a window message box

use: message text $varname$

Example:

eval $a$,1

message The value of a is $a$

***Loop controls***

Here there are some commands to be used for cycling/looping. Please take into consideration that the macro executor shows a confirmation message after 500 instructions executed. This is to avoid infinite loops (watchdog).

**for**

For executes cycles over a variable for the number of cycles indicated. For must be always followed by **next**.

use: for $varname$ number\_of\_cycles\_to\_perform

A fixed value for cycles to perform could be substituted with a variable

use: for $varname$ $num\_cycles$

Example1:

for $a$ 10

message cycle number $a$

next

Example2:

ask $cycles$,for,Enter how many cycles to perform,2,2,20

for $a$ $cycles$

message cycle number $a$ of $cycles$

next

**next**

The next statement must be always connected to a **for**. It is possible to nest cycles.

Example:

for $x$ 2

for $y$ 3

message This message will be showed 6 times

next

next

**label**

Label creates a label used as an entry point for **jump** and **if** statements. The label name must not contain spaces.

Use:

label labelname\_without\_spaces

Example:

label cycle1

**jump**

Jump is an unconditioned jump to a label.

Use:

label labelname

;the code here is executed forever, but the watchdog will ask to continue or to stop

jump labelname

**if**

*If* jumps to the label if the variable is true. Use eventually an eval command for calculating the value of the variable used.

Example:

label loop

ask $a$,IF,Insert 1 to loop,0,0,1

eval $a$,$a$==1

if $a$ loop