PLC Programs

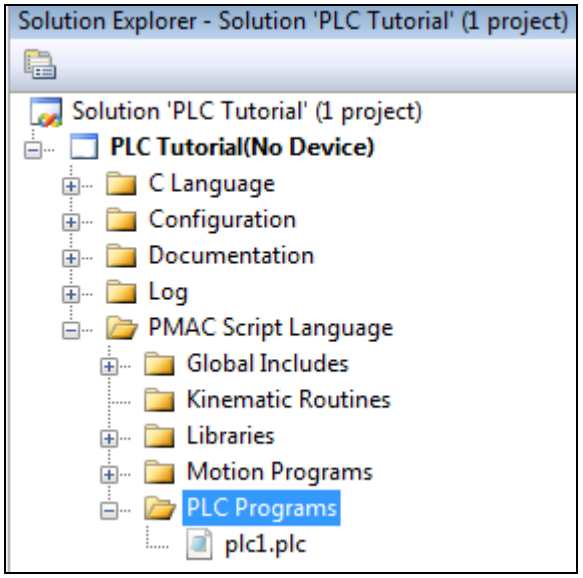
Syntax and Examples

PLC Overview

- **PLC programs (a.k.a. PLCs) are for general purpose use**
  - I/O processing
  - Data gathering
  - Safety checking (limits, current output, etc.)
  - State machine control
  - Starting/stopping motion programs
  - Jogging/homing motors
  - Gain scheduling
  - Sending messages to host PC
- **Written with the same flow control syntax as motion programs**
  - **If/Else** statements
  - **While** loops
  - **Switch** statements
  - Subprogram/subroutine calls through **call**, **gosub**, and **goto** commands
- **The execution sequencing is different from motion programs**
  - Execution does not pause on move commands
  - Execution is asynchronous to running motion programs

PLC Overview

- PLCs are stored under PMAC Script Language→Global Includes→PLC Programs in the Power PMAC IDE Solution Explorer:



PLC Overview

- **Basic Structure:**

```
open plc 1
// Program contents
close
```
Power PMAC Script

- **In Power PMAC, you have the choice of either numbering your PLC with integers (e.g. 1, 2, 3) like above, or with names:**

```
open plc Startup
// Program contents
close
```
Power PMAC Script

- The IDE automatically assigns an internal number corresponding to this named program, starting at 1. You can use it anywhere when starting (with the **enable plc** command) or listing the program's contents (with the **list plc** command).

PLC Overview

- **Up to 32 PLC programs, numbered 0 to 31**
- **Starting from number 0, up to 4 programs can run in foreground at the Real-Time Interrupt rate**
  - **Sys.MaxRtPlc** specifies the highest-numbered PLC to run in real-time; PLC0 is always real-time
  - Changes made to **Sys.MaxRtPlc** only take effect when affected PLCs are disabled
- **Sys.RtIntPeriod sets RTI frequency (every Sys.RtIntPeriod + 1 servo interrupts)**
- **PLCs 4 – 31 always execute in background**
- **PLCs repeat automatically until disabled; no need to "keep alive" with a loop**
- **Key Commands:**
  - **enable plc *i*** // Enables PLC *i*
  - **disable plc *i*** // Disables PLC *i*
  - **list plc *i*** // Lists contents of PLC *i*

  You can also enable or disable multiple PLCs on the same line; for example:

  enable plc 1..5, 7, 31 // Enables PLCs 1 through 5, and 7, and 31
  disable plc 4, 8, 10..15 // Disables PLCs 4, 8, and 10 through 15

- **Can check execution status with PLC[*i*].Active, or IDE Task Manager**

PLC Execution Structure

- Execution of an active PLC is automatically started at appropriate time in real-time interrupt (RTI) or background cycle if it is enabled
- "Enable PLC" sets an internal flag that is checked when the PLC has its next "turn" to run
- Every background cycle, one background PLC runs, then the next background PLC the next background cycle, etc.; all foreground PLCs run every RTI
- Execution continues until end of program or end of (true) while loop – constitutes end of one "scan"
- Next scan does not start until next RTI or next turn in background cycle
- Next scan starts at top of program (if previous scan got to end), or at top of while loop (if previous scan exited at bottom of loop)
- If PLC program commands motion (e.g. jog, homing, or axis move), program execution does not stop as motion program does
  - Must monitor in user code for end of move
- No need to place a PLC within while loop to cause continued scans; Power PMAC will automatically call it repeatedly
  - For a "one-shot" PLC, last line of program should be **disable plc *n***

Operators and Comparators

- **Mathematical Operators:**
  - + (addition)
  - − (subtraction)
  - * (multiplication)
  - / (division)
  - % (modulo, remainder)
  - & (bit-by-bit AND)
  - | (bit-by-bit OR)
  - ^ (bit-by-bit XOR)
  - ~ (bit-by-bit inversion)
  - << (shift left)
  - >> (shift right)
- **Logical Operators:**
  - || (logical OR)
  - && (logical AND)
- **Assignment Operators:**

  Simple assignment: = (expression value written into variable)

  Assignments with arithmetic operation: +=, -=, *=, /=, %=

  Assignments with logical operation: &=, |=, ^=

  Assignments with shift operation: >>=, <<=

  Increment/decrement assignments: ++, --
- **Comparators:**
  - == (equal to)
  - > (greater than)
  - < (less than)
  - ~ (approximately equal to [within 0.5])
  - != (not equal to)
  - <= (less than or equal to)
  - >= (greater than or equal to)
  - ! (not)

Scalar Functions

- Trig functions using radians: **sin**, **cos**, **tan**, **sincos**
- Inverse trig functions using radians: **asin**, **acos**, **atan**, **atan2**
- Trig functions using degrees: **sind**, **cosd**, **tand**, **sincosd**
- Inverse trig functions using degrees: **asind**, **acosd**, **atand**, **atan2d**
- Hyperbolic trig functions: **sinh**, **cosh**, **tanh**
- Inverse hyperbolic trig functions: **asinh**, **acosh**, **atanh**
- Log/exponent functions: **log** (or **ln**), **log2**, **log10**, **exp**, **exp2**, **pow**
- Root functions: **sqrt**, **cbrt**, **qrrt**, **qnrt**
- Rounding/truncation functions: **int**, **rint**, **floor**, **ceil**
- Random number generation : **rnd** (32-bit), **randx** (64-bit), **seed**
- Miscellaneous functions: **abs**, **sgn**, **rem**, **madd** (multiplication & addition), **isnan**

My First PLC

- **Write a PLC to increment a global variable (P-Variable). Example:**

```
global Counter = 0;
open plc increment
Counter++;
close
```

Power PMAC Script

- **Type enable plc increment in the Terminal Window**
- **Put the global variable in Watch Window and watch it increment**

Watch: Online[192.168.0.201:SSH]

	Command	Response
0	Counter	32
*		

while

- **while(*condition*){*contents*}**
  - Performs {***contents***} until ***condition*** goes false
  - Logical condition syntax is C-like
  - Leave {***contents***} blank to wait without performing additional actions
  - If {***contents***} occupies only a single statement, its surrounding brackets ({ and }) may be omitted

- **Example:**

```
while(Input1 == 0) {} // Pause here until Machine Input 1 goes high
while(Input2 == 1)
{
        Counter++; // Increment Counter while Input2 is 1
}
```

Power PMAC Script

Note

Waiting in an empty loop will not cause loss of synchronicity with a master signal.
This example assumes that Input1, Input2, and Counter are previously defined variables.

if

- **if(*condition*){*contents1*} else {*contents2*}**
  - Performs {***contents1***} if ***condition*** is true; otherwise, performs {***contents2***}
  - **else** clause is optional
  - Logical condition syntax is C-like
  - If {***contents1***} or {***contents2***} occupy only a single statement, their surrounding brackets ({ and }) may be omitted

- **Example:**

```
if(Input1 == 0) // If Machine Input 1 is low
{
        Output1 = 0; // Set Output 1 low
} else
{
        Output1 = 1; // Set Output 1 high
}
```

Power PMAC Script

Note

The above example assumes that Input1 and Output1 are previously defined variables.

switch

- **switch(*Variable*){*contents*}**
  - Compares ***Variable*** to a number of distinct, integer (ONLY) states and takes actions for each value. Syntax is C-like.
  - If ***Variable*** matches one of the states listed, that branch of code is executed
  - **break** prevents code execution from passing to subsequent states; omit **break** if the program should continue to subsequent branches
  - The **default** branch of code (see below) executes if ***Variable*** does not match any specified states
- **Example:**

```
switch(MachineState)
{
        case 0:
                // action1
                break;

        case 1:
                // action2
                break;

        default:
                // action3
                break;
}
```

Power PMAC Script

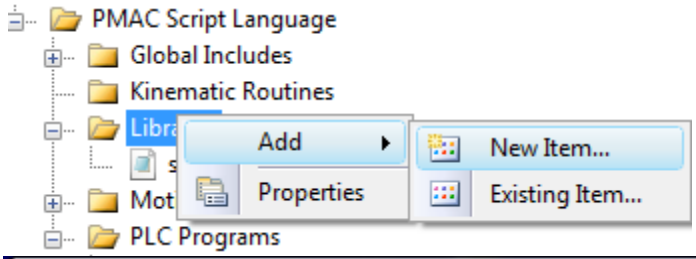
Note

This example assumes that MachineState is a previously defined variable.

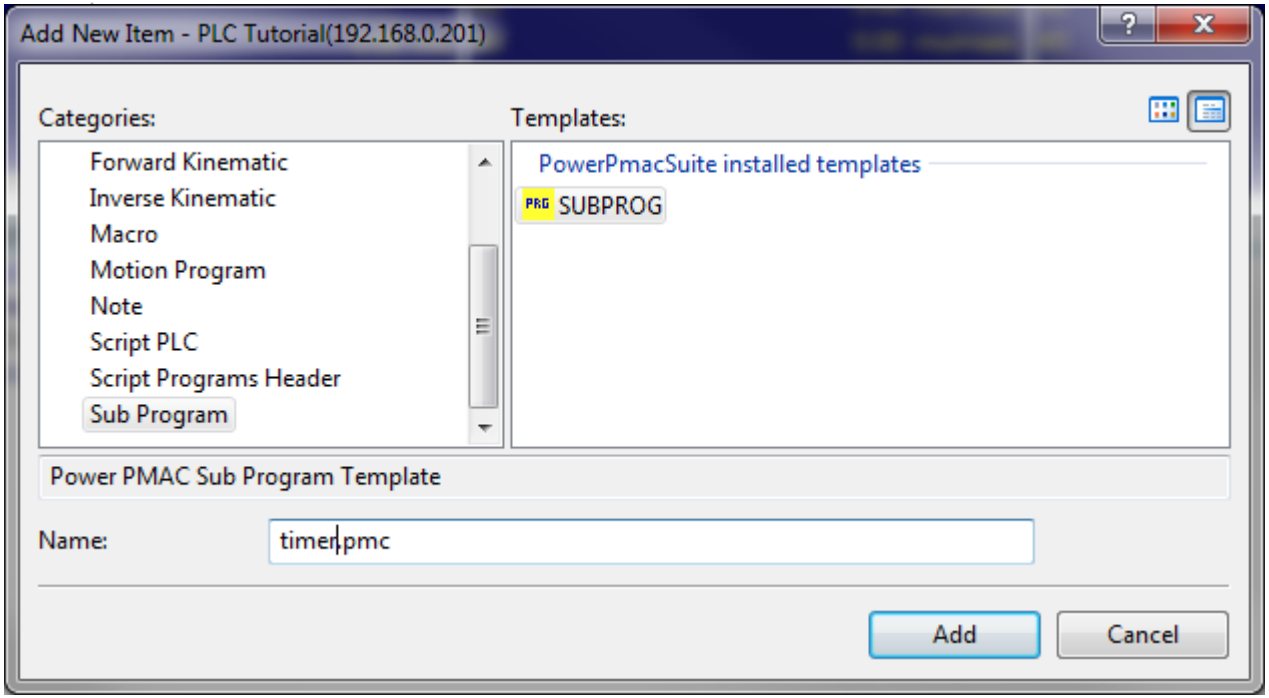
Writing a Delay in a PLC

- In order to create a delay in a PLC, we need to make a "Timer" subprogram
- Open the Global Includes→Libraries folder in the IDE, right click Libraries and choose New Item…, and then Sub Program, and name it "timer.pmc"

1.



2.



Writing a Delay in a PLC

- **In "timer.pmc" we need to make a while loop that waits a specified duration:**

```
open subprog timer(duration)
local EndTime = Sys.Time + duration; // "local" variable to store end time
while(Sys.Time < EndTime){}
close
```

Power PMAC Script

- **Then, in your PLC, call the timer with an argument in units of seconds as shown in this example:**

```
call timer(0.25); // Wait 0.25 seconds before proceeding
```

Power PMAC Script

To learn more about subprograms and subroutines, see the associated tutorial or training slides.

Edge Triggered vs. Level Triggered

- **Level Triggered Example:**

```
open plc leveltriggered
if(Input1==1) {                 // If machine input 1 is high
                Output1=1;      // Activate machine output 1
}
else {
                Output1=0;      // Deactivate machine output 1
}
close
```

Power PMAC Script

- **Edge Triggered (Latching) Example:**

```
open plc edgetriggered
local Latch1 = Input1; // Set initial latch state equal to initial input state
while(1){
if(Input1==1){                  // If machine input 1 is high
                if(Latch1==0){              // If machine input 1 was previously low
                  Output1=1;                // Activate machine output 1
                  Latch1=1;                 // Latch internal machine input 1 signal
                }}
else {if(Latch1==1){            // If machine input 1 is low && previous switch state was high
                  Output1=0;                // Deactivate machine output 1
                  Latch1=0;                 // Delatch internal machine input 1 signal
                }}}
close
```

Power PMAC Script

One should always use Edge Triggered logic when sending commands for jogging, homing, or anything that causes motion. This prevents the command from being dangerously sent repeatedly.

Jogging in a PLC

Command	Example	Description
jog+*[{list}]*	jog+ jog+ 1..5, 8	Jogs the motor(s) indicated in the positive direction. If no motor number, jog last-addressed motor.
jog-*[{list}]*	jog-1,2,3;	Jog negative indefinitely
jog/*[{list}]*	jog/1,2,3;	Closes the loop on indicated motor(s) or stops the motor(s)

jog*[{list}]*=*{data}*
jog*[{list}]*=*
	jog2=3000;
	jog3=(Q1);
	jog7=*
		Jogs motor(s) to specified position. If * indicated, jogs to **Motor[*x*].ProgJogPos**

jog*[{list}]*:*{data}* **jog*[{list}]*:***	jog1..3,5..7:0; jog25..27:*	Jog specified distance (or ProgJogPos if * used) relative to present commanded position
jog*[{list}]*^*{data}* **jog*[{list}]*^***	jog^5000; jog1^(Q1);	Jog specified distance relative to present actual position
jogret*[{list}]*	jogret1,2,3;	Return to pre-jog position
jogret*[{list}]*=*{data}* **jogret*[{list}]*=***	jogret1,2,3=-2468; jogret25..27=*	Jogs to ***{data}*** and sets that as pre-jog position or to ProgJogPos if * used

{jog command}^{data}	jog=10000^-50;
	jog1:-50000^(P10);
	jog1..3=*^0;
		Jog-until-trigger (see homing tutorial)

Homing/Killing in a PLC

➢ Homing commands in a PLC

Home *n* // Homes motor *n* or a list of motors
Homez *n* // Home-zero for motor *n* or a list of motors (sets this position as zero)

Examples:

```
home; // Home presently addressed motor
home1; // Home Motor 1
homez1,2,3; // Zero-move home of Motors 1, 2, & 3
home1..3,5..7; // Home Motors 1, 2, 3, 5, 6, 7
```

Power PMAC Script

➢ Kill command in a PLC

Kill *n* // Kills (removes power from) motor *n* or a list of motors

Examples:

```
kill; // Acts on presently addressed motor
kill1; // Acts on Motor 1, regardless of addressed
kill1,2,3;
kill1..3,5..7;
```

Power PMAC Script

Waiting for the Jog/Home to Finish

- **When using a PLC to jog a motor, one can wait for the jog to finish before advancing in the PLC by polling this parameter until it becomes 1:**

  Motor[*x*].InPos // Motor "in position" status bit

  This becomes 1 when the motor's desired velocity is 0, and the motor is within **Motor[*x*].InPosBand** motor counts for **Motor[*x*].InPosTime** servo cycles.

Motor[*x*].InPosBand is by default 0, which is a strict requirement; you may want to increase this slightly or you might not see Motor[*x*].InPos become 1.

- **When using a PLC to home a motor, one should poll these parameters until they both become 1 before advancing:**

  Motor[*x*].InPos // Motor "in position" status bit
  Motor[*x*].HomeComplete // Motor "home complete" status bit

Synchronous Variables

- **If you are jogging or homing, you can use synchronous variable assignments much like motion programs**
- **Variable is assigned at the same time the jog or home move begins**
- **Assignment types:**

- Simple assignment: == (expression value written into variable)
  - For any type of variable format
- Assignments with arithmetic operation: +==, -==, *==, /==, %==
  - *==, /==, %== for floating-point variables/elements only
- Assignments with logical operation: &==, |==, ^==
  - For integer variables/elements only
- Increment/decrement assignments: ++=, --=

Example:

```
P1=0; P1==1; jog1=100000;                    // P1 set  =1 at start of jog move
while (!(P1) || !(Motor[1].InPos)) { }       // Move not started or ongoing
M1=1;                    // Set output when move has finished and settled
```

Power PMAC Script

Note

If you want to use synchronous variables with a motor in a PLC, that motor must be assigned to a coordinate system.

Starting/Aborting Motion Programs

➢ Starting Motion Programs

start *n*:*m* // Starts program ***m*** in coordinate system ***n***

Note that all motors in the coordinate system must be closed loop in order for this command to work.

Example:

```
start 5:13;     // Start program 13 in coordinate system  5
start 1..3:5; // Start program 5 in coordinate systems 1 through 3
```

Power PMAC Script

➢ Aborting Motion Programs

abort *m* // Aborts coordinate system ***m***

An abort stops any motion program running in the coordinate system and brings the motors in that coordinate system to a controlled stop.

Example:

```
// Abort coordinate system 10
abort 10;
```

Power PMAC Script

Example: Homing and Jogging Motor #1

- **This example homes motor 1, waits for it to finish, and then jogs motor 1 to 2000 cts absolute**

```
open plc jog_home
home 1;                          // Home motor 1
call timer(0.01);                // Wait a small period to force the command to execute
while(Motor[1].InPos == 0 || Motor[1].HomeComplete == 0){} // Wait to finish
jog1=2000;    // Jog motor 1 to 2000 cts
call timer(0.01);                // Wait a small period to force the command to execute
while(Motor[1].InPos == 0){} // Wait to finish
disable plc jog_home
close
```

Power PMAC Script

Example: Jogging PLC with I/O

- **This example will jog the Motor #1 forward if machine input 1 is high, and closed-loop stop the motor when low.**

```
open plc jog_io
Latch1 = Input1;
while(1)
{
        if(Input1==1) // Machine input 1 is high
        {
          if(Latch1==0) // Machine input 1 latch was low
          {
                Latch1=1;     // Bring machine input 1 latch high
                jog+1;        // Jog forward
          }
        }
        else                  // Machine input 1 is low
        {
          if(Latch1==1) // Machine input 1 was high
          {
                Latch1=0;     // Bring machine input latch low
                jog/1;        // Stop jogging
          }
        }
}
close
```

Power PMAC Script

Note

One should always use edge-triggered logic for jogging and homing in PLCs, which is what this PLC example uses.

Time to Practice

- **Exercise 1: Write a PLC to read the machine input switches (inputs 1-8) from input M-Variables on your demo rack and activate an LED in response while the switch is closed using the corresponding output M-Variables for outputs 1-8.**
  - The most efficient way to do this is with a **while** loop that indexes through two arrays of **ptr** variables: one for inputs, one for outputs.

- **Exercise 2: Write a PLC to jog motor 1 forward (jog+) while one of the tactile switches is held, backward (jog-) while another is held, to closed-loop stop when released (jog/).**

- **Exercise 3: Write a PLC that uses the timer to turn on and off lights at a timed interval of your choice. Use the timer subprogram we made earlier and call it with the syntax call timer(duration).**

- **Exercise 4: Write a PLC that will home motor 2 (home 2), check for the home to finish (while(Motor[2].InPos == 0 || Motor[2].HomeComplete == 0){}), and then jog that motor to 5000 counts (jog2=5000), check for it to stop (while(Motor[2].InPos==0){}), and then disable the PLC. Note that you may need to widen Motor[2].InPosBand a little for the InPos check to return 1 in the presence of poor tuning.**

Exercise 1 Solution

- **First, assign pointers to digital I/O:**

```
ptr Inputs(8)->*;
Inputs(0)->GateIo[0].DataReg[0].0.1;
Inputs(1)->GateIo[0].DataReg[0].1.1;
Inputs(2)->GateIo[0].DataReg[0].2.1;
Inputs(3)->GateIo[0].DataReg[0].3.1;
Inputs(4)->GateIo[0].DataReg[0].4.1;
Inputs(5)->GateIo[0].DataReg[0].5.1;
Inputs(6)->GateIo[0].DataReg[0].6.1;
Inputs(7)->GateIo[0].DataReg[0].7.1;

ptr Outputs(8)->*;
Outputs(0)->GateIo[0].DataReg[3].0.1;
Outputs(1)->GateIo[0].DataReg[3].1.1;
Outputs(2)->GateIo[0].DataReg[3].2.1;
Outputs(3)->GateIo[0].DataReg[3].3.1;
Outputs(4)->GateIo[0].DataReg[3].4.1;
Outputs(5)->GateIo[0].DataReg[3].5.1;
Outputs(6)->GateIo[0].DataReg[3].6.1;
Outputs(7)->GateIo[0].DataReg[3].7.1;
```

Power PMAC Script

Note

This assumes the user has a ACC-65E or ACC-68E digital I/O card. Different products may have different digital I/O mappings and you can see the individual products' manuals for that.

Exercise 1 Solution

- **Next, write the PLC:**

```
open plc exercise_1
local index;    // Loop counter and array index
local Latches(8);               // Input latches

index = 0;      // Initialize counter
while(index < 8){               // Loop through all latches
                Latches(index) = Inputs(index);                 // Latch initial input states
                index++;                        // Increment index
}

while(1){       // Keep loop alive now that it is initialized
                index = 0;      // Initialize loop counter
                while(index < 8){               // Loop through all inputs
                                if(Inputs(index) && !(Latches(index))){     // If the input is high but the latch low
                                                Outputs(index) = 1;                     // Activate output
                                                Latches(index) = 1;                     // Latch input
                                }
                                else {          // If the input is low but latch high
                                                Outputs(index) = 0;                     // Deactivate output
                                                Latches(index) = 0;                     // Delatch input
                                }
                                index++;        // Increment index
                }
}
close
```

Power PMAC Script

Exercise 2 Solution

```
open plc exercise_2
local index;
local Latches(4);
index = 0;
while(index < 4){ // Initialize input latches
                Latches(index) = Inputs(index);
                index++;
}

while(1){ // Keep loop alive once latches are initialized
                if(Inputs(0) && !(Latches(0)))                {
                                jog+1;          // Jog mtr1 positive, set this latch and clear others
                                Latches(0) = 1; Latches(1) = 0; Latches(2) = 0; Latches(3) = 0;
                } else
                if(Inputs(1) && !(Latches(1)))                {
                                jog-1;          // Jog mtr 1 negative, set this latch and clear others
                                Latches(1) = 1; Latches(0) = 0; Latches(2) = 0; Latches(3) = 0;
                } else
                if(Inputs(2) && !(Latches(2)))                {
                                home 1;         // Home mtr 1, set this latch and clear others
                                Latches(2) = 1; Latches(1) = 0; Latches(0) = 0; Latches(3) = 0;
                } else if(Inputs(0) == 0 && Inputs(1) == 0 && Inputs(2) == 0 && Latches(3) == 0){
                                jog/ 1;         // Jog stop mtr 1, set this latch and clear others
                                Latches(3) = 1; Latches(1) = 0; Latches(2) = 0; Latches(0) = 0;
                }
}
close
```

Power PMAC Script

Exercise 3 Solution

```
open plc exercise_3
Outputs(0) = 1;          // On
call timer(0.5);         // Wait 0.5 seconds
Outputs(0) = 0;          // Off
call timer(0.5);         // Wait 0.5 seconds
close
```

Power PMAC Script

Exercise 4 Solution

```
open plc PLC_Exercise_4
Motor[2].InPosBand = 0.5; // Widen position band
home 2;        // Initiate the home on motor 2
call Timer(0.01); // Wait a short period to force the home to start
// Wait for the home to finish
while(Motor[2].InPos == 0 && Motor[2].HomeComplete == 0){}
jog2=5000; // Start the jog
call Timer(0.01); // Wait a short period for the jog to start
while(Motor[2].InPos == 0){} // Wait for the jog to finish
disable plc PLC_Exercise_4
close
```

Power PMAC Script