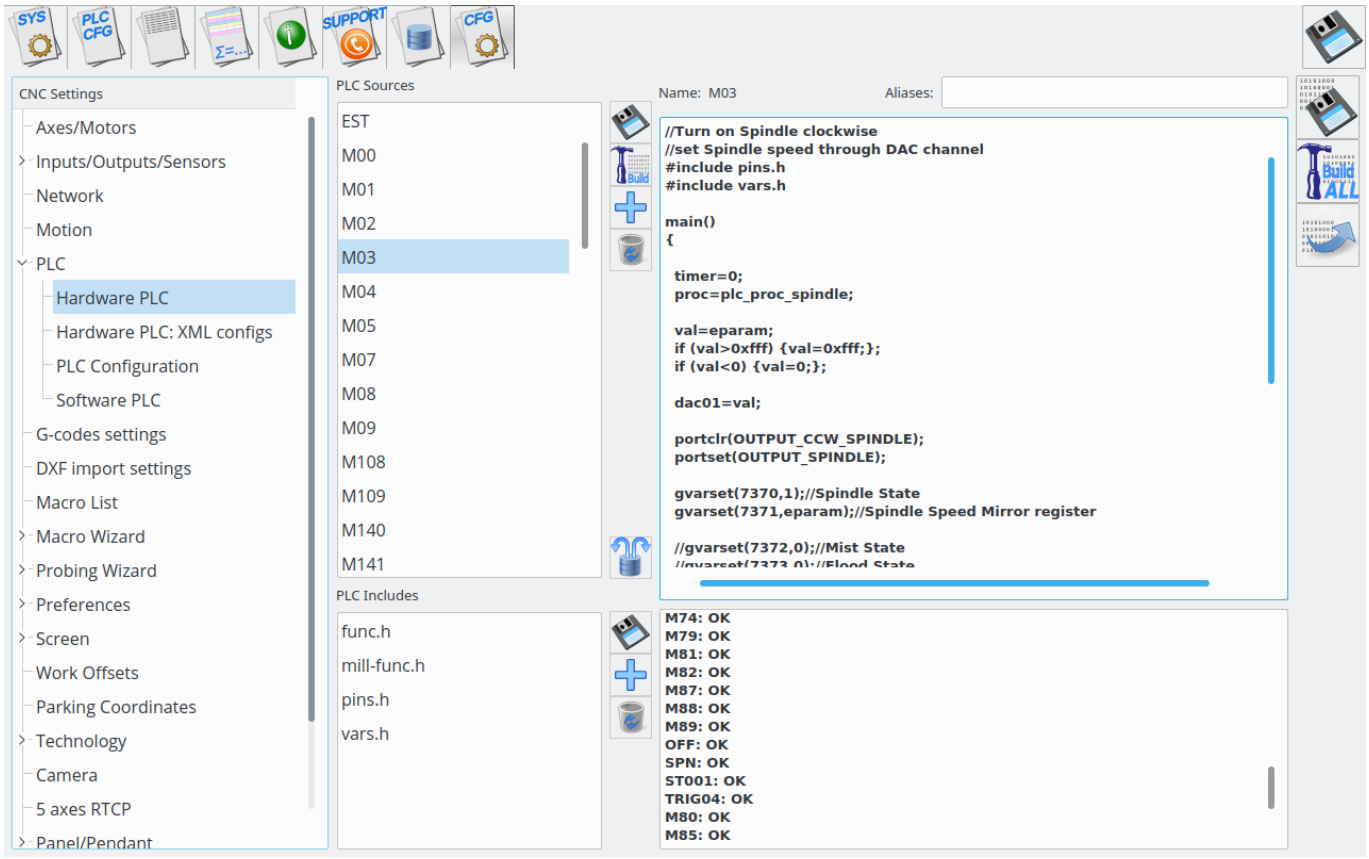
PLC

PLC stands for “Programmable Logic Controller”.

PLC controller can load and run small binary programs (PLC procedures). PLC procedure should be written in a simplified C-like language, compiled and stored in PLC controller memory to be ready to run. myCNC software includes PLC Builder - tiny IDE (Integrated Development Environment) to create and modify PLC procedure source files, compile them to binary code and upload it as RomFS iso image disk to PLC controller memory.



myCNC control has 2 types of built-in PLC systems that named Hardware PLC and Software PLC.

Note that hardware PLC can be launched from software PLC if necessary, through the use of *gvarset(100040,HARDWAREPLC);* command (replace *HARDWAREPLC* with your M-command of choice). You can read more on the process in the Software PLC section down below.

Hardware PLC

“Hardware PLC” means PLC system runs inside the CNC control board and able to access directly to CNC controller peripherals (inputs, outputs, PWMs, DACs, ADCs etc). PLC has also API to access to **Motion Controller** of a myCNC control board, so positioning commands are possible from Hardware PLC.

Hardware PLC is a tiny virtual machine that runs pre-compiled PLC procedures. PLC procedure can be started from

- G-code program M-code
- On-screen button
- Input Pin
- From TCP Ethernet Socket through Server API

Hardware PLC loop cycle time is 1ms. PLC core runs PLC procedure until the end of next loop. At the end of each loop PLC sleeps for 1ms, then continue running PLC. N-times repeat loop (even empty) will be executed N milliseconds. For example, delay for 10ms can be programmes as

```
timer=10; do{timer--;}while(timer>0);
```

```
IMPORTANT.
Hardware PLC is single-tasking. Only one PLC process can be executed at the
moment.
If new PLC procedure loaded, previous PLC process will be terminated
immediately
and replaced by the new one.
```

PLC Language

- PLC operates with 32 bits integer values only. Floating point operations don't work in PLC (typically need to use coefficient conversions to allow for more granular control).
- There is no need to declare variables. There is very limited space of 32 elements for variables.
- There a number of pre-defined variables in the PLC
  - eparam - External Parameter variable. A short manual is available here: Eparam
  - **proc** - variable is used to identify a process running in the PLC.
    The value is sent to the CNC control software and can be used to display Current PLC state (like Idle, Ignition (for plasma cutting), preheat (gas cutting), Tool change, Probing etc)
    - The **proc** variable can impact many behaviours of the myCNC software, such as jogging speed (using the User Settings value for when the **proc** is Idle, and using the cutting speed for **Plasma** and **Cutting** states).
  - **timer** - value can be used as time counter inside PLC procedure
  - vexit - variable contains additional exit code. In case of error CNC control software can display Error message depends on **vexit** exit code.
  - **message**. The message variable is handled every loop delay time. If **message** value is not “0” PLC controller sends to CNC control software message code and reset the variable to “0”.
  - pwm01 - variable mapped to PWM #0 register. Writing to this variable will change PWM #0 value.
  - pwm02 - variable mapped to PWM #1 register. Writing to this variable will change PWM #1 value.
  - pwm03 - variable mapped to PWM #2 register. Writing to this variable will change PWM #2 value.
  - pwm04 - variable mapped to PWM #3 register. Writing to this variable will change PWM #3 value.
  - dac01 - variable mapped to DAC #0 register. Writing to this variable will change DAC #0 value.
  - dac02 - variable mapped to DAC #1 register. Writing to this variable will change DAC #1 value.

- adc01 - variable mapped to ADC #0 register. Reading this variable will return ADC #0 value.
- adc02 - variable mapped to ADC #1 register. Reading this variable will return ADC #1 value. Variables adc01, adc02 are a bit obsolete. It's better to use **gvarget**, **gvarset** functions and The hardware access registers to access to the control board peripherals.

For example, changing the value of PWM01 can be done using the following PLC procedure:

```
main()
{
   val=eparam;
   if (val>0xfff) {val=0xfff;};
   if (val<0) {val=0;};
   pwm01=val;
   exit(99);    //normal exit
};
```

```
PLC is a tiny virtual machine with a very limited register and memory space.
PLC was created for very simple Inputs/Outputs manipulation.
Heavy algorithms cannot be handled by PLC.
Please keep in mind this while creating Hardware PLC procedure.
```

- PLC operators

Operator	Description	Example
+	sum	c=a+200;
-	subtract	c=a-150;
*	multiply	c=a*b;
/	divide	c=b/7;
&	logical AND	c=a&7;
|	logical OR	c=b|1;
>>	binary right shift	c=a>>16;
<<	binary left shift	c=1<<n;
==	equal to	if (a==20) .....
!=	not equal to	if (a!=0) .....
>	more than	if (b>0) .....
>=	more or equal than	if (a>=0xf) .....
<	less than	if (c<10) .....
<=	less or equal than	if (a<=b) .....
++	variable post increment	x++;
--	variable post decrement	a--;

- PLC functions
  - g0moveA - send positioning command for PLC to the Motion controller.
  - portset - set to “1” selected output pin
  - portclear - clear to “0” selected output pin
  - portget - return selected input pin state (“0”, “1”)
  - gvarget - return Global Variable Register value. This function sends an inquiry about Global variable value to myCNC control software running on Host PC and waits till reply received. Waiting time can be 20…200ms depends on Host PC Operating System

(Windows, Linux).
- gvarset - set Global Variable Register to given value. This function sends to myCNC control software inquiry to change Global Variable Register to given value. There is a number of “Mapped” register addresses besides of “real” Global Array registers. List of Mapped addresses is shown in a table below.

Address	Description
20000...20100	Print variable value in myCNC control message widget for debugging purpose. Values written to this registers will be printed in myCNC control software in Message widget - this is useful to see values of certain variables directly in the log window as the program is running.

It is possible to access the state of the output via gvarget commands from within the PLC process:

```
a=gvarget(0x400); //OUT0
b=gvarget(0x407); //OUT7
```

Starting from 0x400 to represent OUT0, this is a hexadecimal system that is simple to convert to dotted decimals (through the likes of a simple reference site here). Thus, for example, *gvarget(0x40d);* will return the state of Output #13.

The hardware access registers

Name	Address	Description
GVAR_HW_INPUTS0	7180	
GVAR_HW_INPUTS1	7181	
GVAR_HW_INPUTS2	7182	
GVAR_HW_INPUTS3	7183	
GVAR_HW_OUTPUTS0	7184	YouTube tutorial
GVAR_HW_OUTPUTS1	7185	
GVAR_HW_OUTPUTS2	7186	
GVAR_HW_OUTPUTS3	7187	
GVAR_HW_INPUTS4	7188	
GVAR_HW_INPUTS5	7189	
GVAR_HW_INPUTS6	7190	
GVAR_HW_INPUTS7	7191	
GVAR_HW_OUTPUTS4	7192	
GVAR_HW_OUTPUTS5	7193	
GVAR_HW_OUTPUTS6	7194	
GVAR_HW_OUTPUTS7	7195	
GVAR_HW_ADC0	7196	
GVAR_HW_ADC1	7197	
GVAR_HW_ADC2	7198	
GVAR_HW_ADC3	7199	
GVAR_HW_ADC4	7200	
GVAR_HW_ADC5	7201	
GVAR_HW_ADC6	7202	
GVAR_HW_ADC7	7203	

Name	Address	Description
GVAR_HW_DAC0	7270	
GVAR_HW_DAC1	7271	
GVAR_HW_DAC2	7272	
GVAR_HW_DAC3	7273	
GVAR_HW_DAC4	7274	
GVAR_HW_DAC5	7275	
GVAR_HW_DAC6	7276	
GVAR_HW_DAC7	7277	
GVAR_HW_PWM0	7278	
GVAR_HW_PWM1	7279	
GVAR_HW_PWM2	7280	
GVAR_HW_PWM3	7281	
GVAR_HW_PWM4	7282	
GVAR_HW_PWM5	7283	
GVAR_HW_PWM6	7284	
GVAR_HW_PWM7	7285	

- Running Motion commands from PLC
- Hardware PLC Examples
  - Getting a Height Map
  - M07 Mist Coolant ON
  - M03 Simple Spindle ON procedure
  - M88 M89 Stop Motion from PLC if Input pin activated
- Gas Cutting Control implementation
- API to work with Modbus devices from PLC
- THC API

Software PLC

“Software PLC” means PLC system runs inside the myCNC software and controls CNC controllers peripherals through Software API.

The main advantage of Software PLC is multitasking. All PLC procedures are running simultaneously and independent from each other.

Software PLC cycle time is 100ms, so Software PLC is suitable for wide range of **slow** applications like automatic lubricant control, fume exhaust control, alarm sensors control etc.

- All Software PLC procedures (except “system” procedures) are compiled and started simultaneously in separate threads with myCNC software start or after “Build All” button pressed in **PLC Builder/Software PLC**.
- “System” PLC procedures are procedures with names start with

  ```
  __ (double underscore)
  ```

  symbols. “System” procedures are not started automatically with the myCNC software start, but instead can be started automatically with some events or manually. There are a few pre-defined “System” PLC handlers-

Handler Name	Comments
`__HANDLER_INIT`	The procedure executed just after myCNC software started AND configuration is sent to the myCNC control board. The procedure can be used to perform some inputs testing and switch outputs just after the software started.
`__HANDLER_EXIT`	The procedure executed just before the myCNC software closed.
`__HANDLER_GCODE_START`	The procedure executed just before the myCNC software run g-code (after pressing PLAY button)
`__HANDLER_GCODE_STOP`	The procedure executed just after the myCNC software finished g-code running (after pressing STOP/PAUSE button)
`__BV17`	Initially named BV17, this Software PLC had double underscores added to remove an issue with automatic enabling of the testing mode for the controller peripherals (by becoming a System PLC, it prevents the procedure from starting automatically when the program is loaded).

Variables used in Software PLC:

Variable	Use	Example	Comment
100020	Jog the selected axis (100020 through to 100027)	*gvarset(100020, 100);*	Negative values are not accepted, use 0-X (for example, *gvarset(100020, 0-100);*)
100040	Launch a Hardware PLC from within a Software PLC	*gvarset(100040, 607)*;	This will launch Hardware PLC M607
100041	Eparameter to feed into the Hardware PLC being launched using 100040	*gvarset(100041, 333);*	Used before *gvarset(100040, 607);*, this will set eparam=333

- Software PLC Examples
  - How to add mandatory Homing after Emergency Button and-or Servo ready alarm
  - Button to toggle select output pin with indication
  - Oil Change counter
  - Controller Peripherals Test - BV17
  - Charge Pump

Compare Software and hardware PLC

Parameter	Software PLC	Hardware PLC
Loop time	100ms	1ms
Hardware access	Slow, through the software and Ethernet communication	Fast, direct in myCNC controller
Multitasking	Yes, all PLC procedures are running simultaneously in separate threads	No, starting new procedure will terminate a current procedure
Code size for PLC procedure	16k bytes	512 bytes
Total disk size for PLC procedures	Unlimited	8k bytes

Language Core for Software and Hardware PLC are almost the same so language syntax is pretty the same and similar to C-language but VERY VERY stripped down.

PLC Language

PLC variables

- **“32”** - is a total number of variables PLC supports
- No need to define variables in source text. Variable name reserved when fist used. For example in line

```
output0=15;
```

PLC builder will define variable “output0” and assign the value to “15”

PLC predefined variables

- **var00**...**var15** : This variables are initialised before PLC procedure start from values defined in plc-variables.xml file.
- **proc** : proc variable value can be displayed on myCNC screen with display name “display-plc-proc” and can be used to show current status of PLC procedure
- **message** : If message value set to non-zero value, PLC controller will send a message to myCNC Control Core or myCNC software with parameters defined in variables
  message,
  var00 (usually defined as **command** as well) and
  var01 (defined as **parameter**).
  This way PLC controller able to communicate with Motion Controller and Host Software.
  Message variable is handled and cleared by PLC controller while “loop” operation, so **message** variable assign usually followed by short timeout loop like

```
timer=2;do{timer--;}while(timer>0);
```

Examples:

```
//send to myCNC software information spindle speed is "0"
command=PLC_MESSAGE_SPINDLE_SPEED_CHANGED;
parameter=0;
message=PLCCMD_REPLY_TO_MYCNC;
timer=2;do{timer++;}while (timer>0);
```

```
//Probing for Plasma Cutting.
g0moveA(0x0,0x4,0-30000);                    //Start move down
do { code=gvarget(6060); }while(code!=0);  //Wait till Motion
Controller confirm moving started
do{
 code=gvarget(6060);                //check Motion Controller (MC)
current status
 sens=portget(INPUT_IHC);           //and check Probe input
 if (sens==0)
 {
   message=PLCCMD_LINE_STOP;        //Send to MC command to Stop current
```

```
line and wait next command
   code=1;                          //Set flag to exit from the loop
 };
}while (code==0);                   //Exit from loop is motion finished
OR Probe sensor pressed
```

PLC defines

Name	Value	Comment
PLCCMD_MOTION_CONTINUE	1001	
PLCCMD_MOTION_SKIP	1002	
PLCCMD_MOTION_SOFT_SKIP	1003	
PLCCMD_MOTION_PAUSE	1004	
PLCCMD_PLC_PAUSE	1005	
PLCCMD_PLC_FORK_PAUSE	1006	
PLCCMD_LINE_SOFT_STOP	1007	
PLCCMD_LINE_STOP	1008	
PLCCMD_REPLY_TO_MYCNC	1100	
PLCCMD_SET_CNC_VAR	1010	
PLCCMD_SET_THC_VAR	1011	
PLCCMD_SET_DEVICE_VAR16	1012	
PLCCMD_SET_DEVICE_VAR32	1014	
PLCCMD_SET_CNC_EXTVAR	1020	
PLCCMD_MOTION_ABORT	1032	
PLCCMD_MOTION_BREAK	1033	
PLCCMD_SAVE_POS	1040	
PLCCMD_THC_START	1050	
PLCCMD_THC_STOP	1051	
PLCCMD_THC_PAUSE	1052	
PLCCMD_THC_CONTINUE	1053	
PLCCMD_WATCHBIT1_ON	1060	
PLCCMD_WATCHBIT2_ON	1061	
PLCCMD_WATCHBIT3_ON	1062	
PLCCMD_WATCHBIT4_ON	1063	
PLCCMD_TRIGGER1_ON	1060	
PLCCMD_TRIGGER2_ON	1061	
PLCCMD_TRIGGER3_ON	1062	
PLCCMD_TRIGGER4_ON	1063	
PLCCMD_TRIGGER1_OFF	1064	
PLCCMD_TRIGGER2_OFF	1065	
PLCCMD_TRIGGER3_OFF	1066	
PLCCMD_TRIGGER4_OFF	1067	
PLCCMD_WATCHBIT1_OFF	1064	
PLCCMD_WATCHBIT2_OFF	1065	
PLCCMD_WATCHBIT3_OFF	1066	
PLCCMD_WATCHBIT4_OFF	1067	

PLCCMD_WATCHBIT5_ON	1068	
PLCCMD_WATCHBIT6_ON	1069	
PLCCMD_WATCHBIT7_ON	1070	
PLCCMD_WATCHBIT8_ON	1071	
PLCCMD_WATCHBIT5_OFF	1072	
PLCCMD_WATCHBIT6_OFF	1073	
PLCCMD_WATCHBIT7_OFF	1074	
PLCCMD_WATCHBIT8_OFF	1075	
PLCCMD_WAIT_FOR_CAMERA	1090	
PLCCMD_WAIT_VARSET	1091	
PLCCMD_WAIT_VARCLEAR	1092	
PLCCMD_CAMERA_START	1093	
PLCCMD_CAMERA_FINISH	1094	
PLCCMD_PLC_DEBUG	1098	
PLCCMD_PLC_RESTART	1099	
PLC_BROADCAST_INQUIRY0	1200	
PLC_BROADCAST_INQUIRY1	1201	
PLC_BROADCAST_INQUIRY2	1202	
PLC_BROADCAST_INQUIRY3	1203	
PLC_BROADCAST_INQUIRY4	1204	
PLC_BROADCAST_INQUIRY5	1205	
PLC_BROADCAST_INQUIRY6	1206	
PLC_BROADCAST_INQUIRY7	1207	
PLC_BROADCAST_INQUIRY8	1208	
PLC_BROADCAST_INQUIRY9	1209	
PLC_BROADCAST_INQUIRY10	1210	
PLC_BROADCAST_INQUIRY11	1211	
PLC_BROADCAST_INQUIRY12	1212	
PLC_BROADCAST_INQUIRY13	1213	
PLC_BROADCAST_INQUIRY14	1214	
PLC_BROADCAST_INQUIRY15	1215	
PLC_BROADCAST_OK	1220	
PLC_BROADCAST_ABORTED	1221	
PLCCMD_MODBUS_SPINDLE_SPEED	1230	
PLCCMD_MODBUS_SPINDLE_CMD	1231	
PLCCMD_SET_PIDTIME	1240	

Note on PLC define naming

Please note that there exist some limitations of the preprocessor that parses the *#define* lines. Specifically, it is not possible to have a full name of one parameter be part of the name of another parameter.

For example, an argument name such as *OUTPUT_SPINDLE* (present by default in the **pins.h** file), means that there must be no other *define*-names containing this substring. As such, names such as

```
OUTPUT_SPINDLE_COOL
```

```
OUTPUT_SPINDLE_BEARING
OUTPUT_SPINDLE_CONE
```

are NOT allowed, ssince they all contain the string “OUTPUT_SPINDLE”. Instead, you can make argument names such as

```
OUTPUT_BEARING_SPINDLE
```

or

```
OUTPUT_CONE_SPINDLE
```

since those do not contain the exact string from before. Failure to properly define your arguments in such a manner and then utilizing them in your PLC commands will result in a error during compilation with a label *“syntax error, unexpected ID”*.

PLCCMD_MOTION_CONTINUE and PLCCMD_MOTION_SKIP

The abovementioned PLCCMD_MOTION_CONTINUE and PLCCMD_MOTION_SKIP commands are highly useful for certain applications since typically the motion controller runs commands one by one. By design, if within a running program the next code is a PLC M-code, then the movement will be stopped and the controller will run the PLC program. A message from within the PLC called PLCCMD_MOTION_CONTINUE is used to instruct the Motion controller to read and run the next code from the buffer (thus starting the next motion command).

After this code, the PLC procedure continues running through its code while at the same time the next motion code is launched. In this way, both the PLC procedure and the motion command will be running simultaneously.

This is useful for applications such as homing since it makes it possible to both move the axis and monitor the home sensor in the PLC procedure at the same time. That way when the sensor is activated, the current movement command will need to be stopped. A different message called PLCCMD_MOTION_SKIP is then used - the motion controller will cancel current motion (it will stop moving) and will read the next code from the buffer.

PLC processes named

PLC process name	Value	Comment
plc_proc_check_plasma	10	
plc_proc_venting	11	
plc_proc_start_power	12	
plc_proc_cooling	14	
plc_proc_plasma	15	
plc_proc_wait_plasma	18	
plc_proc_pierce	27	
plc_proc_no_plasma	62	
plc_proc_check_preflow	16	
plc_proc_check_cutflow	17	

plc_proc_check_gases	23	
plc_proc_test_out	24	
proc_m_function	30	
proc_zeroing	32	
plc_proc_probing	33	
plc_proc_ignition	50	
plc_proc_preheat	51	
plc_proc_soft_start	52	
plc_proc_piercing	53	
plc_proc_flame	54	
plc_proc_cutting	60	
plc_proc_purge	61	
plc_proc_no_cutting	62	
plc_proc_moveup	65	
plc_proc_spindle	70	
plc_proc_idle	0	

PLC exit codes list.

Normally PLC procedure should return code 99.

```
exit(99);
```

In case Error happened PLC procedure may return an error code. MyCNC software will catch exit code and report about Error if the error code is in PLC exit codes list.

Exit code name	Value	Comment
plc_process	0	exit code is zero when PLC process is not finished yet
plc_exit_gas_fail	2	PLC aborted, No air pressure sensor
plc_exit_plasma_timeout	3	PLC aborted, No Arc sensor signal Timeout
plc_exit_plasma_fail	4	PLC aborted, PLC Plasma Start cutting procedure error
plc_exit_alarm_key	5	PLC aborted, Emergency key pressed
plc_exit_coolant_fail	6	PLC aborted, No Coolant flow sensor
plc_exit_probe_error	7	PLC aborted, No signal from probe sensor
plc_exit_motor_shorted	8	~~PLC aborted, Motor short circuit detected (ET2/ET4 boards)~~
plc_exit_broadcast_error	10	Error send broadcast message in multi-device configuration
plc_exit_normal	99	Normal exit
plc_exit_extern_break	100	PLC procedure aborted from outside of PLC core

PLC messages

Message name	Value	Comment
PLC_MESSAGE_PLASMA_OK	101	
PLC_MESSAGE_WATCHBIT_ACTION	110	
PLC_MESSAGE_SPINDLE_SPEED_CHANGED	120	
PLC_MESSAGE_PULL_OUT_TOOL	130	
PLC_MESSAGE_SPINDLE_TURNING 131		

PLC_MESSAGE_ENCODER_DATA	140	
PLC_MESSAGE_GVAR_VALUE	141	
PLC_MESSAGE_UID	142	
PLC_MESSAGE_HCONTROL_OFFSET	144	
PLC_MESSAGE_TANGENT_ANGLE	145	
PLC_MESSAGE_USER	146	
PLC_MESSAGE_HCONTROL_JSPEED	147	
PLC_MESSAGE_HCONTROL_DC	148	
PLC_MESSAGE_ASK_RECALC	150	
PLC_MESSAGE_SOFTLIMIT_STOP	151	
PLC_MESSAGE_GVARSET	153	
PLC_MESSAGE_IHC_NOT_CONNECTED	155	
PLC_MESSAGE_IHC_ERROR	156	
PLC_MESSAGE_LATHE_GEARS	160	
PLC_MESSAGE_ERR_VM	-1	
PLC_MESSAGE_ERR_ROMFS	-2	
PLC_MESSAGE_MOTION_BUFFER_EMPTY	-5	
PLC_MESSAGE_ERR_SENSOR_COOLANT	-10	
PLC_MESSAGE_ERR_SENSOR_AIR	-11	
PLC_MESSAGE_ERR_SENSOR_GAS	-12	
PLC_MESSAGE_ERR_SENSOR_OXYGEN	-13	
PLC_MESSAGE_ERR_SENSOR_PLASMA	-14	
PLC_MESSAGE_ERR_PROBING	-15	
PLC_MESSAGE_ERR_PLASMA_FAIL	-20	
PLC_MESSAGE_ERR_PLASMA_TIMEOUT	-21	
PLC_MESSAGE_PRESSED_ALARM_KEY	-30	
PLC_MESSAGE_PRESSED_STOP_KEY	-31	
PLC_MESSAGE_MOTOR_SHORTED	-35	
MAPPED_OUT_THC_LOWSPEED	63	
MAPPED_OUT_LOW_MOTOR_CURRRENT	65	

Controller peripherals API from PLC procedures

A number of Global variable addresses are mapped to Hardware Inputs/Outputs. PLC procedure can access the controller peripherals through GVarGet/GVarSet function. Addresses to access to controller hardware are listed below

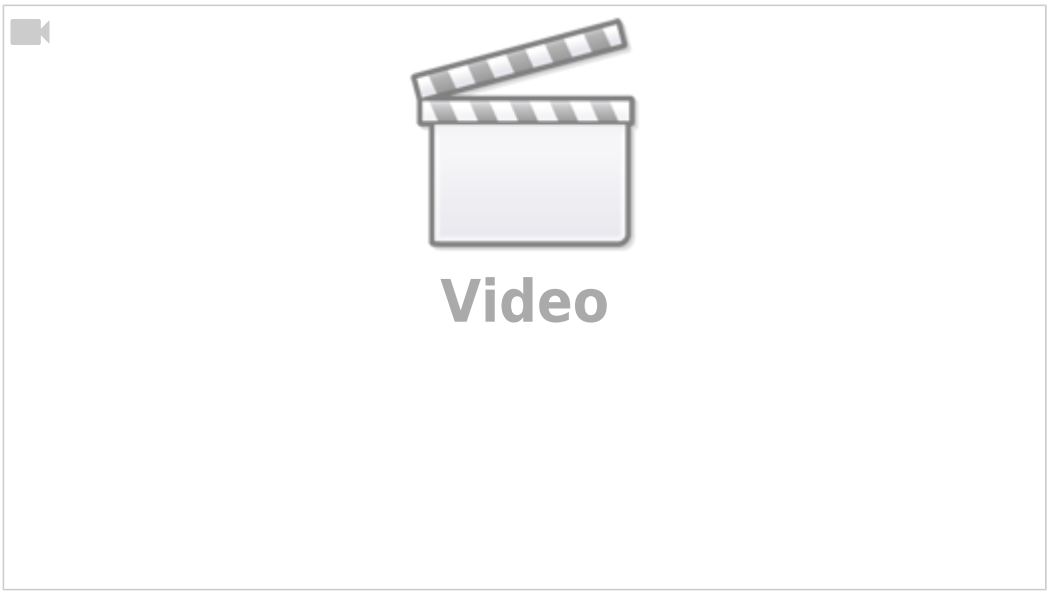
Variabe Name	Address	Description
GVAR_HW_INPUTS0... GVAR_HW_INPUTS3	7180... 7183	
GVAR_HW_OUTPUTS0... GVAR_HW_OUTPUTS3	7184... 7187	
GVAR_HW_INPUTS4... GVAR_HW_INPUTS7	7188... 7191	
GVAR_HW_OUTPUTS4... GVAR_HW_OUTPUTS7	7192... 7195	

Variabe Name	Address	Description
GVAR_HW_ADC0… GVAR_HW_ADC7	7196… 7203	
GVAR_HW_DAC0… GVAR_HW_DAC7	7270… 7277	
GVAR_HW_PWM0… GVAR_HW_PWM7	7270… 7277	
GVAR_ET5_ENCODER …		
GVAR_ET5_ENCODER_Z …		
GVAR_ET5_ENCODER_WZ …		
GVAR_ET5_ENCODER …		
GVAR_MODBUS_READ		
GVAR_THC0_CONTROL	7570	
GVAR_THC1_CONTROL	7575	
GVAR_CURRENT_MOTION_CODE	6060	
GVAR_MD_MASTER_MOTION_CODE	7140	
GVAR_CURRENT_TOOL_NUMBER	5400	
GVAR_HCONTROL2_VREF		
GVAR_PLC_MOVE_PROCESS		
GVAR_CAMERA_READY	7090	
GVAR_CURRENT_MACHINE_POSITION	5021, 5022, 5023, 5024, 5025, 5026	
GVAR_CURRENT_PROGRAM_POSITION	5041, 5042, 5043, 5044, 5045, 5046	
GVAR_ENCODER_Z_EVENT		
—	17001	Return Current PROGRAM X Position in PLC units (0.01mm)
—	17002	Return Current PROGRAM Y Position in PLC units (0.01mm)
—	17003	Return Current PROGRAM Z Position in PLC units (0.01mm)
—	17004	Return Current PROGRAM A Position in PLC units (0.01degree)
—	17005	Return Current PROGRAM B Position in PLC units (0.01degree)
—	17006	Return Current PROGRAM C Position in PLC units (0.01degree)
—	17007	Return Current PROGRAM U Position in PLC units (0.01mm)
—	17008	Return Current PROGRAM V Position in PLC units (0.01mm)
—	17009	Return Current PROGRAM W Position in PLC units (0.01mm)
—	17021	Return Current MACHINE X Position in PLC units (0.01mm)
—	17022	Return Current MACHINE Y Position in PLC units (0.01mm)

Variabe Name	Address	Description
—	17023	Return Current MACHINE Z Position in PLC units (0.01mm)
—	17024	Return Current MACHINE A Position in PLC units (0.01degree)
—	17025	Return Current MACHINE B Position in PLC units (0.01degree)
—	17026	Return Current MACHINE C Position in PLC units (0.01degree)
—	17027	Return Current MACHINE U Position in PLC units (0.01mm)
—	17028	Return Current MACHINE V Position in PLC units (0.01mm)
—	17029	Return Current MACHINE W Position in PLC units (0.01mm)

Launching a PLC command using an on-screen button

The following video illustrates the process of creating a button to launch a software PLC command (a similar process can also be used for hardware PLCs):



Launching a Hardware PLC procedure from Software PLC

It is possible to launch Hardware PLC procedures from within a Software PLC command. This can be done for purposes such as utilizing certain commands are not available from the Software PLC, and that some require low-latency sensor monitoring. It also has the added benefit of utilizing the unlimited number of procedures that can work simultaneously in Software PLC, allowing the user create things such as permanent while loops, etc.

For example, a code such as:

```
main()
{

```

```
gvarset(100040,602);

exit(99);

};
```

will launch Hardware PLC M602 from the specified Software PLC.

Additionally, the Param variable can also be used to call a Hardware PLC with a certain eparam variable.

For example, adding a line such as

```
gvarset(100041, Param);
```

will launch the M602 Hardware PLC with the Param variable.

Jog from PLC

***NOTE**: Jog from PLC requires a firmware update. As of February 2022, the feature is available in the Testing firmware branch.*

myCNC allows the user to call for a jog command from within PLCs. The advantage of this motion mode is that it allows to perform tasks (such as changing the speed and direction of movement, as well as turning the motion for a particular axis on or off), all without stopping. This is useful in such applications as homing along multiple (for example, three) axes. Such an example (with simultaneous positioning along the X and Y axes) is available here: PLC Examples

In this mode, the acceleration is set via the following command:

```
gvarset(8631,100); //acceleration time 100ms = 0.1s
```

while the speed is set via the global variable #8634 (the axis mask is stored in the high-order byte):

```
gvarset(8634,((1<<24)|1000)); //Jog speed for X
gvarset(8634,((2<<24)|1000)); //Jog speed for Y
gvarset(8634,((4<<24)|1000)); //Jog speed for Z
```

A sample motion code may therefore look the following way:

```
jog()
{
 gvarset(8631,100); //acceleration time 100ms = 0.1s
 gvarset(5539,1); //switch to fast g0moveA implementation
 gvarset(8632,1000); //Jog Speed 1m/min
 gvarset(8634,((1<<24)|1000)); //Jog speed for X
 gvarset(8634,((2<<24)|1000)); //Jog speed for Y
 gvarset(8634,((4<<24)|1000)); //Jog speed for Z

 gvarset(8635,1); //Jog X+
```

```
timer=2000;
do
 {
    timer--;
    if ((timer&0xff)==0)  {    gvarset(8635,1);   };
  }while(timer>0);

gvarset(8634,((1<<24)|500)); //Speed 500 for X
gvarset(8635,1); //Jog X+
timer=2000;
do
 {
    timer--;
    if ((timer&0xff)==0)  {    gvarset(8635,1);   };
  }while(timer>0);

gvarset(8634,((1<<24)|200)); //Speed 200 for X
gvarset(8635,1); //Jog X+
timer=2000;
do
 {
    timer--;
    if ((timer&0xff)==0)  {    gvarset(8635,1);   };
  }while(timer>0);

gvarset(8635,2+1<<8); //Jog X- AND Y+ Simultaneously
timer=2000;
do
 {
    timer--;
    if ((timer&0xff)==0)
    {
      gvarset(8635,2+1<<8); //Jog X- AND Y+ Simultaneously
    };
  }while(timer>0);

  gvarset(8635,0);
  };
```

The following variables may be used for this jog functionality:

Variable name	Variable #
GVAR_G0PLC_SPEED	8630
GVAR_G0PLC_TIME	8631
GVAR_G0PLC_SPEED_UNITS	8632
GVAR_PLC_JOGSPEED_UNITS	8634
GVAR_PLC_JOG	8635

An example of a homing PLC that utilized this functionality is shown below. This PLC is designed for the Z-axis only:

```
#include pins.h

wait_move()
{
    do { code=gvarget(6060); }while(code!=0x4d);  //wait till motion
finished
};

show_error()
{
    gvarset(9121,1); //bring up popup message 21 - this can be customized
    timer=50;do{timer--;}while(timer>0);
    message=PLCCMD_MOTION_BREAK;  //1033
    exit(99);
};

find_home_z()
{

  gvarset(5521,1); //disable hardware limits
  gvarset(5525,1); //disable software limits

  gvarset(8631,50); //acceleration time 100ms = 0.05s

  statez=0; //we set the "triggered" state to 0 by default
  speed=500;
  speed_slow=100; //used on rollback for better precision

  direction=(4<<8); //set the direction variable to Z-

  gvarset(8634,(4<<24)|speed); //Jog speed for Z

  gvarset(8635,direction); //jog in the set direction Z-
  timer=0;
  do
   {
      changed=0;
      if (statez==0) //to home X
       {
          sens=portget(INPUT_HOME_Z); //get state of home x sensor
          if (sens!=0)
          {
            statez=1;
            gvarset(8634,(4<<24)|speed_slow); //Jog speed for Z
            changed=1;
          };
       };
      if (statez==1) //rollback from home Z
       {
          sens=portget(INPUT_HOME_Z); //receive the state of the sensor
          if (sens==0)
```

```
          {
            statez=2;
            changed=1;
          };
       };


  if (changed!=0) //if any of the sensors state is changed
  {
    direction=0; //set direction to 0 (no movement) before flipping
    if (statez==0) { direction=direction | (4<<8);  };  //direction set to
X-
    if (statez==1) { direction=direction | 4;  }; //direction set to X+
    gvarset(8635,direction); //jog in new direction for the axes
  };



      if ((timer&0xff)==0)   {     gvarset(8635,direction);    };
      timer++;
      ready=(statez==2);

    }while(ready==0);


   gvarset(8635,0); wait_move(); //stop jog

};

main()
{

  gvarset(8631,50); //acceleration time 50ms = 0.05s
  gvarset(5539,1); //switch to fast g0moveA implementation

  find_home_z();


  exit(99);  //normal exit
};
```

The PLC above can be combined with the XY homing (linked at the start of this section) to create simultaneous XYZ homing. In general, the Jog from PLC functionality allows for this simultaneous movement, cutting down on the previously required separate procedures per each axis.

For more examples, see Jog from PLC