

GENERAL INFORMATION

This is the on-line help system of the AML compiler (AmlComp) for the AXV brushless drive. You can navigate the help using the menu tree on the left or following the links in the text.

The main steps covered in the help are:

[AML features and the development environment](#)

This sections explains the AML concepts and its main elements, the tools needed for the program development and the typical development path.

[The compiler main elements and its usage](#)

In this part are presented the AML compiler elements and its features.

[The AML language](#)

This section contains all the information about the language, the syntax and a detailed description of each instruction.

[Application notes and program examples](#)

This last section gives some hints on how to write programs and to deal with some application requirements.

THE AML PROGRAMMING ENVIRONMENT

The AML language and its execution environment has been conceived to allow the AXV users to write repetitive positioning sequences.

The way AML allows to reach this goal is quite simple. Programs are written in a language very similar to the CNC programs.

Programs are executed sequentially by the AXV, and so the way AML programs are written is sequential and not cyclical like the PLC. Thus the language concept is close the purpose that has to be achieved.

Another important feature is the use of user units (millimetres, inches, degrees etc.) for specifying the positions instead of encoder counts and so on.

The AML execution isn't however an alternative to PLC programs. PLC remains an element needed by the drive to perform the real time processing and to manage the resources of the drive itself.

Also the AML controller on the drive uses the PLC execution. Some examples are the homing execution and the [macro](#) interfacing.

The relations between the AML controller and the PLC are better depicted in the [AML execution environment](#) part.

The tools

To perform the AML programming with the AXV drive, the following elements are required:

- An AXV drive with the AML feature enabled with the AML key and properly connected with the RS485 cable to the PC.
- The AML compiler AmlComp.
- If the drive is to be configured, the Cockpit configuration program is needed.

- The PLC compiler (GPLC or other IEC-1131 compiler) is requested for special drive customizations or AML macro programming.

Once the drive configuration is performed and the eventual PLC customization is completed, the only tool needed for the AML development is the AmlComp compiler.

The development path

To start with the AML programming, the following steps should be carried out:

- Enable the AML with the activation key.
- Load the AML START default application into the drive **or** develop a specific application with GPLC.
- Configure the drive using Cockpit.
- Start the AML development with AmlComl
- Optionally write the macros into the PLC application.

Consult the [AML concepts](#) to understand how AML programs work.

AML MAIN CONCEPTS

The complete [AML environment structure](#) is shown in the relative page.

Program storing

The AML program is stored into an apposite sector of the FLASH memory of the drive. Thus the program remains permanently into the drive also when is switched off.

When the drive is turned on, the AML program is loaded into the operative RAM memory for the execution.

During development with the compiler, the program is downloaded directly into RAM to allow the immediate execution of the program under construction.

Once the programmer terminates the development and the debugging, the program can be saved into the FLASH memory (with the proper [compiler option](#)) for the subsequent normal program execution.

Into the drive can be stored only one AML program.

Program execution

The AML program, once loaded into the RAM at the start-up, is executed by the dedicated AML controller on the drive.

The execution starts from the first line of the program. The controller executes one line at a time, each command must complete the execution before the next line is executed.

The completion of each instruction (such as positioning, wait, macros etc.) is verified by the controller with a period of 2 ms that is the controller execution period.

The instructions that can be executed immediately (such as assignments) are executed in one controller cycle (2 ms), the following instruction will be executed into the next cycle.

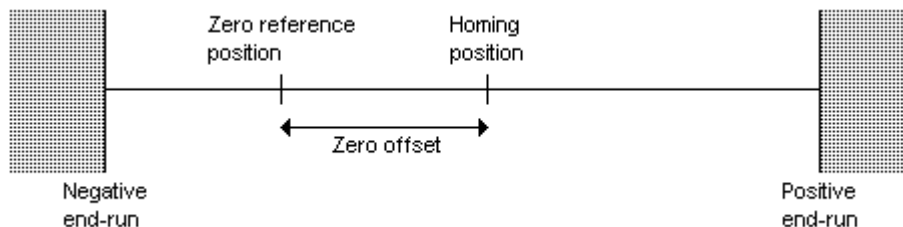
Other AML basic concepts are:

- [The reference system](#)
- [The program states](#)

- [The AML parameters](#)
- [The PLC interface](#)

THE AML REFERENCE SYSTEM

The AML language is based on a reference system like the one shown below:



The main elements of the reference system are:

- **The space measure unit:** the measure unit is chosen by the user as he needs. The [ENCC](#) parameter indicates the distance, in the unit settled, covered with an encoder revolution. In the following descriptions, the measure unit chosen is abbreviated with UU (user unit).
- **The time measure unit:** the time measure unit is fixed for the AML system and is the second. Therefore the speed is expressed into UU/s and the acceleration into UU/s².
- **The end runs:** they represent the limits of the reference system and are indicated by the [QMIN](#) and [QMAX](#) parameters. The AML controller will never move outside these limits. If a position command is given outside these limits, it will not be executed.
- **The homing position:** is the position where the axe performs the homing, normally executed with a digital sensor and the encoder index. In general is different from the zero reference or the position 0.
- **The zero reference position:** is the position along the reference system that corresponds to zero into UU. This position can, in some cases, be outside the end runs.
- **The zero offset:** is stored into the [QZERO](#) parameter. Is expressed into UU and indicates the position of the homing inside the reference system.

The schema refers to a linear axes but AML can also manage [circular axes](#).

EXECUTIVE STATES

AML program controller can undertake these six operative conditions, described in the following section:

NOT ENABLED	AML controller is in this state when AXV drive is not enabled or it is in alarm status, or no AML program is downloaded into the drive. In this state no program can be executed.
NOT READY	This state is active when the drive is enabled, without alarms, but homing hasn't been performed. No AML program can be executed.
	This state is active when the drive is enabled, without alarms, homing

READY	performed and a program has been downloaded into the drive. The AML controller is in this state also when the program has performed STOP or END commands. Only in this state, it is possible of using jog commands to move the motor manually.
RUN	This state consists in program execution. The AML controller enters in this condition after received START commands, when the drive was in READY state.
JOG	This state is active when jog command are sent to the drive for manual positioning and AML controller is in READY state.
ERROR	This state is active when an error occurs during the AML program execution. In this case, the AML controller gets disabled until when a RESET command is sent to or the drive is turned on again.

AML PARAMETERS

AML parameters can be divided in two different categories:

- [System configuration parameters](#);
- [Program parameters](#).

Configuration parameters

AML configuration parameters allow defining system properties, as regard mechanical coupling, speed, accelerations and positioning limits.

SPDNOM	It indicates the nominal speed of the axis. This is the maximum speed that the axis can reach. In the program, it is possible to indicate the speed that has to be used by SPD variables. The nominal speed is expressed in unit of space, used for ENCC, divided by time in seconds.
ACCNOM	It indicates the nominal acceleration of the axis. This is the maximum speed that the axis can support. In the program, it is possible to indicate the acceleration that has to be used by ACC variables. The nominal acceleration is expressed in unit of space, used for ENCC, divided by squared time in seconds.
QZERO	It indicates the position in which the axis performs the homing. The unit of measurement is the same of ENCC parameter one.
ENCC	This parameter defines the space covered by the axis in an encoder round. The unit of measurement is chosen according to necessity (millimeters, meters, grades, etc.). This unit of measurement characterizes the working unit of the whole AML system. Also speeds and accelerations are expressed in the chosen unit of space.
ERRPOS	It indicates the space threshold inside which the axis is considered positioned, once the positioning is ended.
MAXERRPOS	It indicates the maximum dynamic error inside which the axis must stay during AML program execution.

	If this threshold is exceeded, the execution of AML program stops and the system signals the positioning alarm.
QMIN	It indicates the minimum quote that the axis can reach (minimum endrun). If in program a quote less than QMIN is set, the system signals the no-valid position alarm.
QMAX	It indicates the maximum quote that the axis can reach (maximum endrun). If in program a quote greater than QMAX is set, the system signals the no-valid position alarm.
SPDJOG	It indicates the speed during a jog command. It is expressed in percentage of the nominal speed.

AML program parameters

Program parameters contain the values (quotes, timers etc.) that can be used inside AML programs.

Using of program parameters, AML programs can be configured and managed by external controllers like CNCs, keyboards and so on by means of the communication interfaces (RS485, CAN or ProfiBus). Each AML is addressed using its own parameter index (IPA).

The following table summarizes the allowable AML program parameter set.

Type	Name	Description	Number	IPA
Quotes	Q_n	Used to store different positions	50 (Q0 to Q49)	12000÷12049
Timers	T_n	Time values to be used in WAIT commands	50 (T0 to T49)	12100÷12149
Parameters	P_n	General purpose parameters	50 (P0 to P49)	12200÷12249
Flags	F_n	Flags for conditional jumps and I/O assignments	64 (F0 to F63)	12300÷12363

THE PLC INTERFACE

Is very important to understand that there is a very close relation between AML programs and the PLC running on the drive.

This link is obtained by means of the mapping of the AML control variables into the PLC addressable space. Thus PLC program can directly read and write variables used by AML programs.

The PLC program is fundamental to AML environment to perform the following tasks:

- Activate the commands as enable, start, stop, jog etc.
- Execute the homing cycle and pass the home encoder coordinates to AML.
- (Optional) execute the macro cycles.

The default AXV application named [AML START](#) already manages all the basic PLC interface needed by AML.

The following are the variables shared between PLC and AML:

AML control bits				
Name	Address	Type	Access	Description
amlEnable	MX60.0	BOOL	RW	AML program execution enable
amlCIOOk	MX60.1	BOOL	RW	Homing executed flag
amlJogP	MX60.2	BOOL	RW	Positive jog command
amlJogN	MX60.3	BOOL	RW	Negative jog command
amlSingleStep	MX60.4	BOOL	RW	Single step program execution command
amlStart	MX60.5	BOOL	RW	Start program command
amlStop	MX60.6	BOOL	RW	Stop program command
amlResetPrg	MX60.7	BOOL	RW	Reset program command (reinitialize to line 1)
amlReady	MX60.8	BOOL	R	Ready status flag
amlRun	MX60.9	BOOL	R	Run status flag
amlError	MX60.10	BOOL	R	Error status flag
amlDirAssiC	MX60.11	BOOL	R	Circular axes direction flag

AML configuration				
Name	Address	Type	Access	Description
amlVelNom	MW61.0	REAL	RW	Nominal speed into user units
amlAccNom	MW61.1	REAL	RW	Nominal acceleration into user units
amlQuZero	MW61.2	REAL	RW	Homing position into user units
amlAccopEnc	MW61.3	REAL	RW	Encoder coupling
amlErrPos	MW61.4	REAL	RW	Position error threshold
amlMaxErrPos	MW61.5	REAL	RW	Maximum dynamic position error
amlFinecMin	MW61.6	REAL	RW	Negative end run
amlFinecMax	MW61.7	REAL	RW	Positive end run
amlVelJog	MW61.8	REAL	RW	Jog speed into user units
amlCurrPos	MW61.9	REAL	R	Actual position into user units
amlCurrErr	MW61.10	REAL	R	Actual position error into user units
amlCurrSpd	MW61.11	REAL	R	Actual speed into user units
amlCurrAcc	MW61.12	REAL	R	Actual acceleration into user units

The above variables correspond to the [AML system parameters](#).

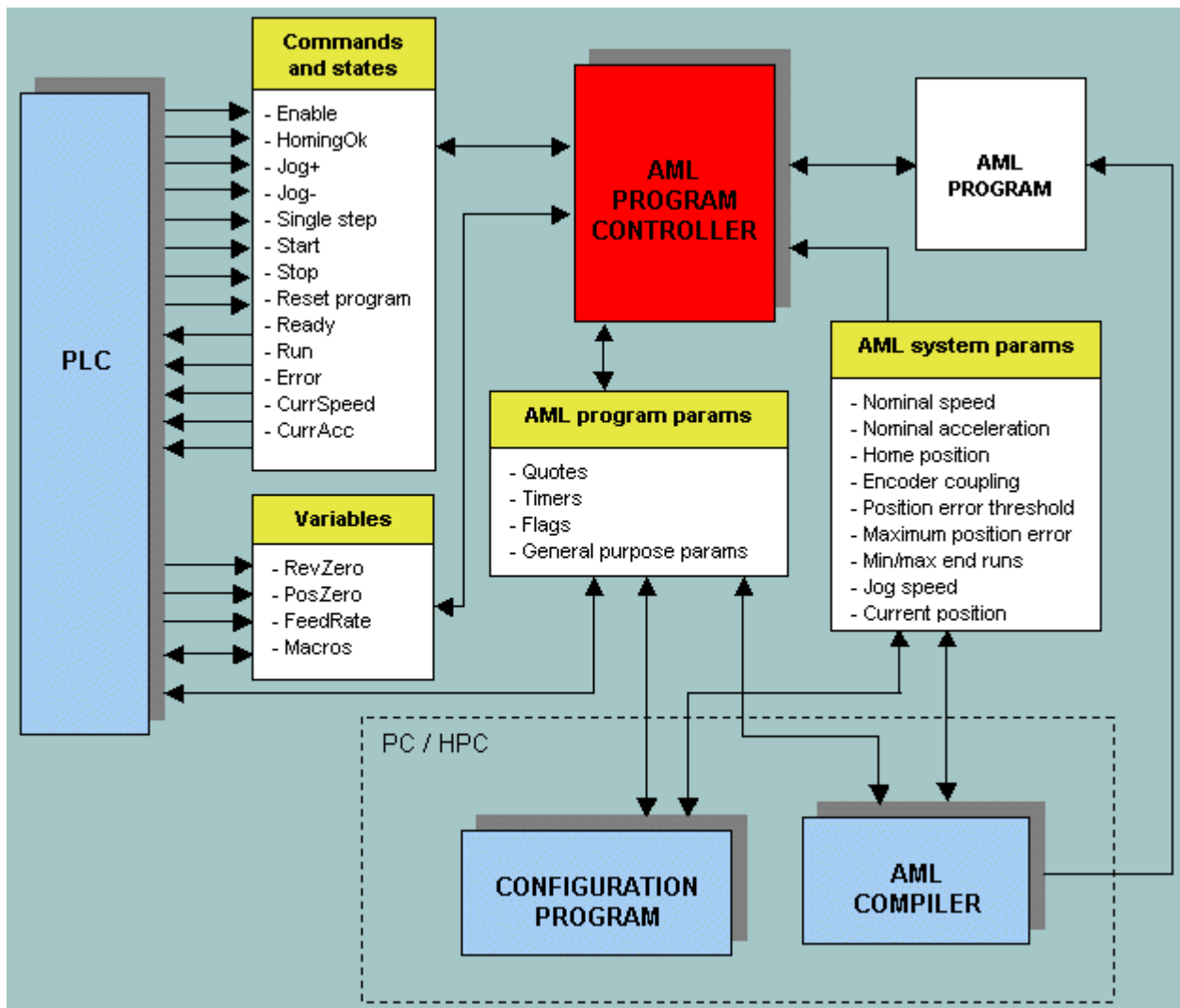
AML control variables				
Name	Address	Type	Access	Description
amlTurZero	MW62.0	DINT	RW	Encoder revolution in the homing position
amlPosZero	MW62.1	DINT	RW	Encoder angular position in the homing position
amlFeedRate	MW62.2	DINT	RW	Feed rate value (from 1 to 100)
amlM0 to amlM63	MW74.0 to MW74.63	BOOL	RW	AML macro invocation flags (these flags are used programming)

AML program variables				
Name	Addresses	Type	Access	Description
amlQuota0 to amlQuota49	MW70.0 to MW70.49	REAL	RW	Quotes AML program parameters
amlTime0 to amlTime49	MW71.0 to MW71.49	REAL	RW	Timers AML program parameters
amlPar0 to amlPar49	MW72.0 to MW72.49	DINT	RW	General purpose AML program parameters
amlFlag0 to amlFlag63	MW73.0 to MW73.63	BOOL	RW	Flag parameters

The above variables correspond to the [AML program parameters](#).

AML EXECUTION ENVIRONMENT

The following picture describes the AML interfaces, aboard AXV drive, with the whole system and with external program tools.



USING THE AML COMPILER

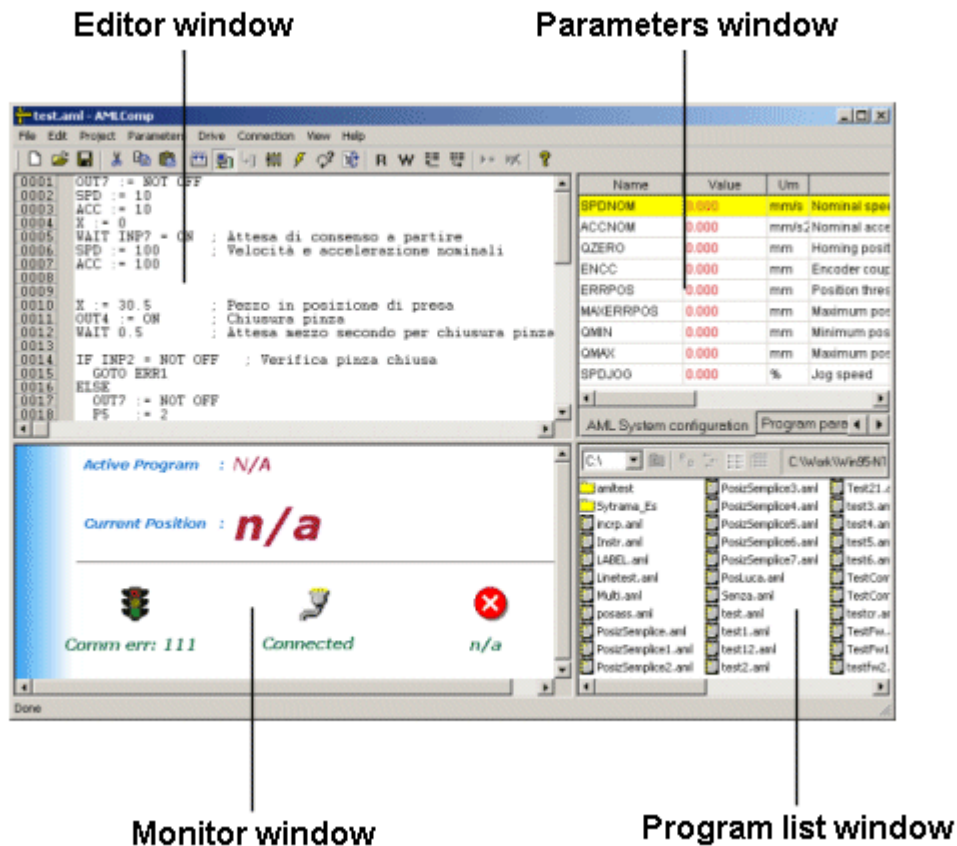
This section explains the features and contains the indications on how to use the AML compiler.

The main steps to perform in order to manage AML applications are the following:

- **Enabling the AML controller:** the AML controller on the drive must be enabled with the proper activation key. If the right key isn't entered, all AML features are disabled. The key is unique for each drive. Contact Phase Motion Control to get the AML key for the drive.
- **AML system configuration:** is the first operative step and it's normally executed once for each drive. It's necessary to set up the [reference system](#) and to indicate the operative limits of the system.
- **Program editing:** here the motion program is conceived and written using the editor. Is part of the editing session also the determination of the [program parameters](#) needed.
- **Program compilation and download:** once the editing is completed, the program has to be compiled and then, if no programming errors are present, it can be downloaded into the drive.
- **Debugging:** the program debugging is executed on-line, using the relative compiler feature. The debugging is normally executed using the single-step mode program execution.

- **Program saving:** when debugging of the program is completed, the program itself can be stored permanently into the drive using the save command of the compiler. The program is then loaded at every drive boot-up and executed when the start command is issued.

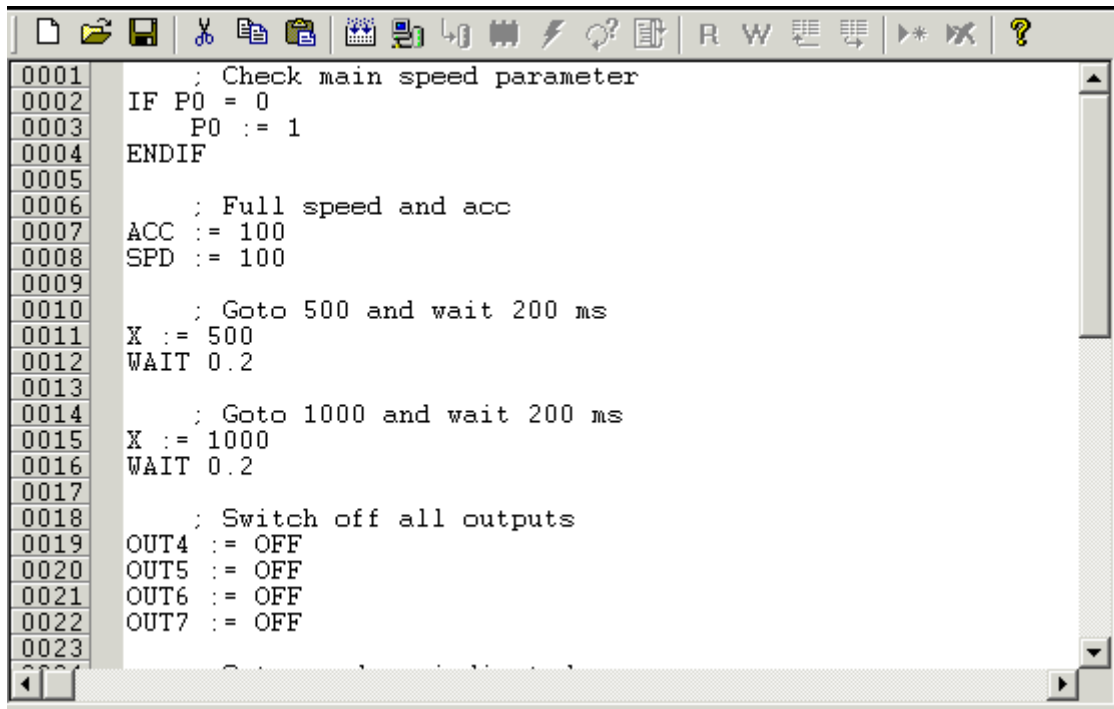
The following figure shows the AML compiler layout with its main windows:



EDITOR WINDOW

The editor window is used to write and modify AML programs. It's organized in lines that are shown on the left size.

On the top of the editor window is placed the compiler toolbar. Each button of the toolbar invokes, when pressed, the relative command of the main menu.



```

0001 ; Check main speed parameter
0002 IF P0 = 0
0003     P0 := 1
0004 ENDIF
0005
0006 ; Full speed and acc
0007 ACC := 100
0008 SPD := 100
0009
0010 ; Goto 500 and wait 200 ms
0011 X := 500
0012 WAIT 0.2
0013
0014 ; Goto 1000 and wait 200 ms
0015 X := 1000
0016 WAIT 0.2
0017
0018 ; Switch off all outputs
0019 OUT4 := OFF
0020 OUT5 := OFF
0021 OUT6 := OFF
0022 OUT7 := OFF
0023

```

To edit an AML program two options are available:

- create a new program using the "File - New" menu option. In this case an empty editor window will be shown.
- open an existing AML program using the "File - Open" menu option. The editor window will display the contents of the program. To open an existing program is also possible to select it from the [program list window](#).

Important: when an AML program is loaded, also its program parameters are loaded. Parameters are shown into the apposite [parameter window](#). AML compiler automatically links a parameter file to each AML source file (with .aml extension). Therefore must be remembered to take also the .ric file (containing program parameters) when, for example, you need to copy the file on a floppy disk.

At the end of the editing process, the file must be saved with the "File - Save" menu option in order to proceed with the compilation and the rest of the developing process.

If the program is new and never saved before, the user will be requested for the name to assign to the new program.

When the compilation is invoked and if the program is not saved, the compiler allows to proceed with saving automatically.

Editing functions

The editing functions of the compiler are the same of the more common Windows editors with Cut, Copy, Paste and Drag & Drop features.

Error report

When the [compilation process](#) is executed, the editor reports the eventual syntax errors by issuing a message box and automatically select the word of the program that generates the error.

```

0001 ; Check main speed parameter
0002 IF P0 = 0
0003     P0 := 1
0004 ENDIF
0005
0006 ; Full speed and acc
0007 ACC := 100
0008 SPD := 100
0009
0010 ; Goto 500 and activate a device
0011 X := 500
0012 WAIT 0.2
0013
0014 ; Goto 1000 and start execution of macro 0
0015 X := 1000 MO
0016 WAIT 0.2
0017
0018 ; Switch off all outputs
0019 OUT4 := OFF
0020 OUT5 := OFF
0021 OUT6 := OFF
0022 OUT7 := OFF

```

IPA	
160	SPD
161	ACC
162	OZEI

AMLComp

Compilation error: Invalid coordinate

Source: C:\Documents and Settings\axel7\Desktop\AmlSamples\Simple.aml
Line : 15

OK

Debugging

When the [debug functions](#) are activated, the editor window change to read-only mode. No program modification are allowed.

During debugging the editor displays also a moving selection bar that indicates the program line that is currently executed by the drive.

The selection bar moves and the editor windows automatically scrolls during program execution.

```

0003
0004 ; Limited speed and acc
0005 ACC := 50
0006 SPD := 50
0007
0008 ; Goto 0 and wait a start cycle command
0009 X := 0
0010 WAIT INP7 = ON
0011
0012 ; Goto 500 and activate a device
0013 X := 500
0014 OUT4 := ON
0015
0016 ; Goto 1000 and start execution of macro 0
0017 X := 1000 MO
0018
0019 ; Goto the beginning
0020 GOTO START
0021
0022

```

Position acquiring

Another function active with the editor window is the [auto acquisition](#) of axes position. Refer to the relative section for more information.

PARAMETERS WINDOW

The parameters windows manages both AML system parameters and program parameters. The appropriate parameter set can be selected using the underlying folder tab on the bottom.

AML system parameters management

The AML system parameters define the [AML reference system](#).
The figure below shows a typical AML system parameters set.

IPA	Name	Value	Um	Description
160	SPDNOM	5000.000	mm/s	Nominal speed
161	ACCNOM	10000.000	mm/s ²	Nominal acceleration
162	QZERO	0.000	mm	Homing position
163	ENCC	100.000	mm	Encoder coupling
164	ERRPOS	0.100	mm	Position threshold
165	MAXERRPOS	5.000	mm	Maximum position error
166	QMIN	-1000.000	mm	Minimum position (negative end run)
167	QMAX	1000.000	mm	Maximum position (positive end run)
168	SPDJOG	100.000	%	Jog speed

AML System configuration Program parameters

The fields in the grid have the following meanings:

Name	Access	Description
IPA	ReadOnly	Indicates the parameter index. The parameter index can be used to address the parameters using serial communication or FieldBus connection.
Value	ReadWrite	Is the actual value of the parameter. If displayed in red, the value shown is the one loaded from the parameter file and may be different from the actual value in the drive. When shown in black, the value shown has been uploaded from the drive.
Um	[ReadOnly]	Is the measure unit for the relative value. The user can modify only the Um corresponding to the ENCC parameter. All displacement units are then displayed with the same string entered for ENCC parameter.
Description	ReadOnly	Explains the parameter meaning.

Managing the system parameters files

When the compiler is started, the last AML system parameters set is automatically loaded into the editor.

If no files was previously loaded, the compiler loads a predefined parameters set that contains the AML [standard system parameters](#).

If desired, the actual parameters set can be saved on the disk in order to store a particular parameter set for the current drive or application. The saving is executed by means of the "Parameters - Save AML parameters file" or "Parameters - Save AML parameters file as ..". The saved file (with the .par extension) can then be loaded with the menu command "Parameters - Open AML parameters file".

Reading and writing parameters of the drive

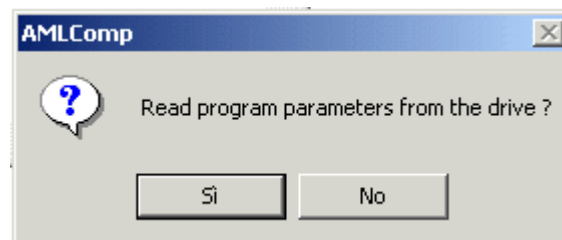
Once the [connection](#) with the drive is established, four commands are available to exchange parameter values with the drive.

- Read single parameter
- Read all parameters
- Write single parameter
- Write all parameters

The single parameter options operate only on the actually selected parameter. All these commands can be issued from the "Parameters" menu or from the relative buttons on the toolbar. Also shortcut keys are available (Ctrl+R, Ctrl+W).

When parameters are read from the drive for the first time their values change color from red to black.

Each time the communication is activated and a program is loaded into the editor, the compiler asks for a complete parameter upload. This feature allows to have an updated set of parameters or to send the parameter set to the drive before proceeding with other operations.



AML program parameters

As explained for the [program editing](#), each program can have its own set of parameters. These parameters allow to change the program behavior by setting their value from external controllers or from the PLC program. The [program parameters](#) meaning is explained into the relative section. Each set of parameters are automatically saved by the compiler together with the AML source file.

The program parameters editor is shown in the figure below:

IPA	Name	Value	Um	Description
12200	P0	5	%	Main movement speed
12100	T0	2	s	End program wait
12000	Q0	-15	mm	Start position

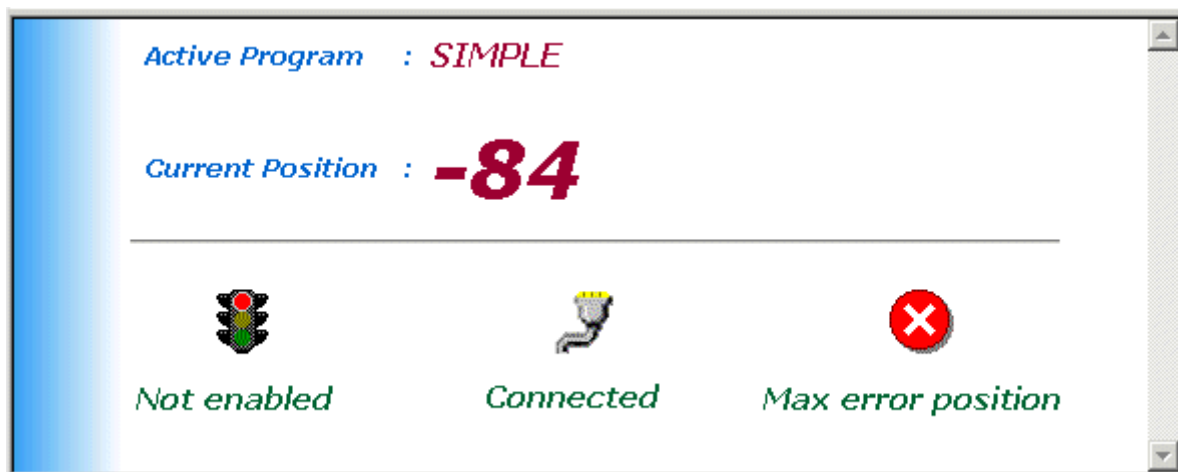
AML System configuration Program parameters

The editor fields have the same meanings as for the AML [system parameters editor](#). The only exception is the possibility to change the contents of the description field. Also the parameters exchange with the drive follows the same rules of the AML system parameters.

The program parameters editor can also receive position values during the [auto acquisition](#) function. Refer to the relative section for more information.







MONITOR WINDOW

The monitor window continuously display the status of the drive.



The fields of the monitor window are (from top to bottom and left to right):

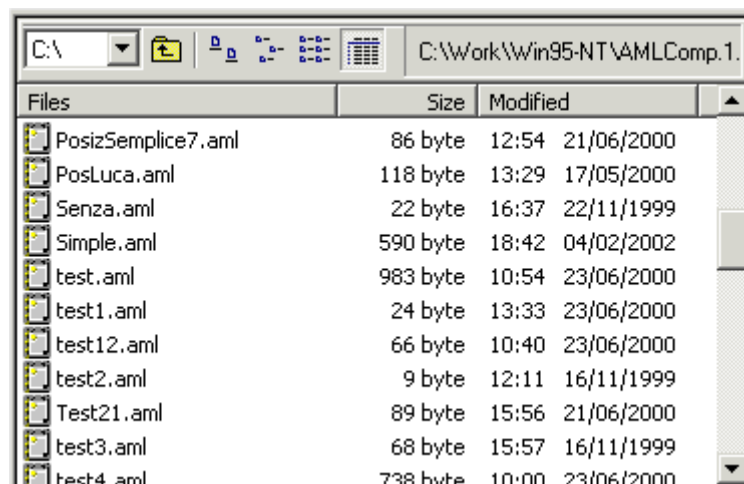
Field	Values	Meaning
Active program		Is the active program in the AML execution memory
Current position		Displays the actual position into user units. Before the homing cycle it always displays 0.
		The status is undefined (normally when the communication does not work)
		AML program is NOT ENABLED state (AML key not enabled or program not enabled)

Executive state		(Blinking) The AML controller is in the NOT READY state.
		(Fixed) The AML controller is in the READY state.
		The AML controller is in the RUN or JOG state.
Connection state		The communication with the drive is not activated.
		The communication with the drive has been activated.
Active error		There is an active program error and the AML controller is in the ERROR state

PROGRAM LIST WINDOW

The program list window shows all the AML programs stored in the selected folder. The folder selection can be done by navigating between the disk folders of the PC.

It's possible to open directly an AML program by simply selecting a source file in the list and then double click on it. This operation gives the same result as with the "File - Open" menu option.



DRIVE CONNECTION

The connection with the drive is activated or closed with the "Drive - Connect" menu option or the relative toolbar button.

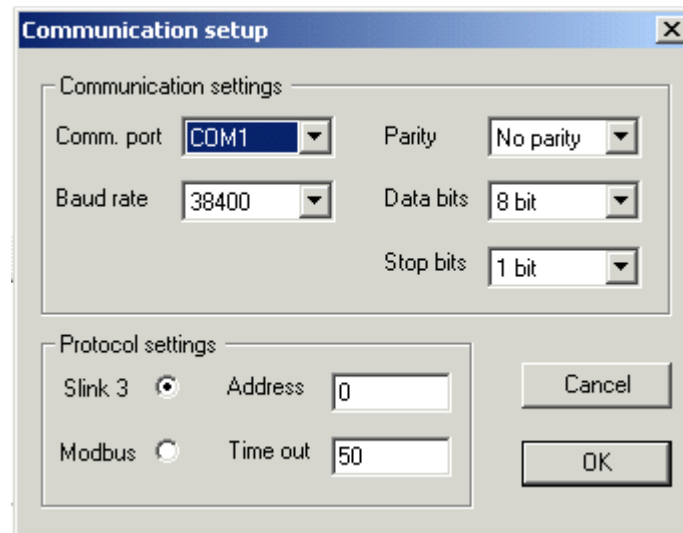
The connection status is displayed into the [monitor window](#).

To properly activate the communication the following steps must be followed:

- Connect the cable to the drive and, if necessary, use an appropriate RS232-RS485 converter (PCI485).

- Set the correct baud rate and line settings; the AXV default values are 38400 baud, No Parity, 8 data bits, 1 stop bit.
- Set the correct drive protocol address.
- Check the connection status in the monitor window (the error code 111 means connection timeout)

All the communication settings can be done activating the "Communication - Settings" menu command. The following window must appear.



PROGRAM DEVELOPMENT

This section contains some procedures normally performed during AML program development.

- [Compilation](#)
- [Program download](#)
- [Debugging](#)
- [Program saving](#)
- [Auto-acquisition](#)

Compilation

The compilation process is started with the "Project - Compile" menu option, the relative button of the toolbar or by pressing the F7 key.

The source file must be saved before compiling. The compiler ask for file saving automatically if the source hasn't been saved before. See also the [editor window](#) topic for details.

Program download

In order to activate the program on the drive, it must be downloaded. The download can be done only when the [communication](#) with the drive is established.

Another condition to be met for program downloading is the **NOT ENABLED** state of the controller. Use also the [monitor window](#) to understand the controller state.

If downloading is performed when the controller is into another state, an apposite message box is displayed by the compiler to inform about this situation.

The download is activated with the "Project - Download" menu option, the relative button of the toolbar or by pressing the F5 key.

Important: remember that after the successful download, the program is stored into the executive

RAM. To store the program permanently, it must be saved.

Debugging

The debugging is activated with the "Project - Debug" menu option.

The debug process is possible only when the program in the editor window has been compiled and downloaded to the drive.

If this condition isn't met, the compiler issues a message that informs the user that the program into the drive is not the same as the on into the editor window or isn't of the same version.

When debugging is active, the editor window displays the apposite indicator bar, see the [editor topic](#) for more details.

During the debugging is very useful to activate the drive Single-Step execution mode in order to execute an instruction at a time.

Program Saving

When the developing process (edit, compile, download and debug) is completed, the program can be saved into the drive FLASH memory. This command is executed by means of the "Drive - Save program" menu option.

Is then possible to reset the drive with the "Drive - Reset" command in order to check that the program is correctly loaded after the start-up.

Auto-acquisition

The auto-acquisition is a feature available during program editing.

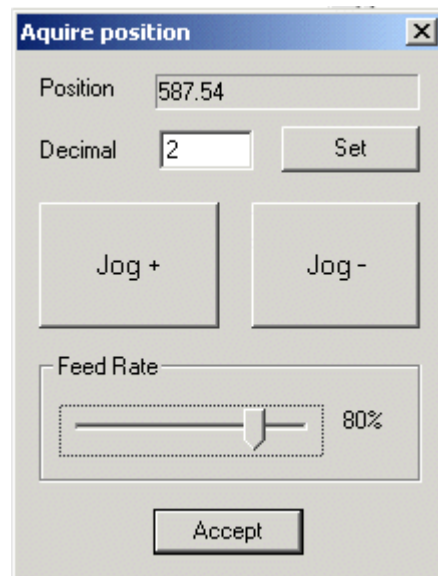
It allows to get the current position of the axe and put it directly into the source editor window or into an apposite Q parameter of the program (see also the [editor window](#) and [parameter window](#) topics).

The function is activated using the "Drive - Acquire position" menu command. The following dialog-box will appear:



The user can choose where the acquired position has to be put. Then appears the auto-acquisition window that displays the current position and allows to specify the number of decimal to use and to execute the jog commands in order to move the axe into the desired position.

When the desired position is reached, the "Accept" button puts the position directly were desired.



THE AML LANGUAGE DESCRIPTION

AML language allows describing positioning sequences, logic operations, I/O interactions that have to be performed by AXV drives.

The AML aim is to allow the definition of cyclic repetitive productions with a language, which has the following main properties:

- Sequential execution of programs (not cyclic execution like PLCs);
- Use of engineering units of measurement;
- Interaction with drive I/O and PLC program;

The Program Structure

AML language programs are made by routines, organized in lines. So, each program line is an operative instruction for the executive module on the AXV drive.

AML controller executes program instructions in succession, starting from the first operative line of downloaded program. Only when the operation of current instruction is finished, AML controller goes through the next line. After that the operation indicated in the last line has been done, AML controller jumps to the first program line again and automatically it resumes execution from start point.

Every line is marked by a number and, sometimes, by a label in order to have a simple logic identification and to allow the execution of jumps.

Operands

AML operands are:

X	Position register: it indicates the position that has to be reached.
SPD	Speed register; it indicates the percentage of the speed used during the positioning (values between 0÷100). Default value is 100.
ACC	Acceleration register: it indicates the percentage of the acceleration used during the positioning (values between 0÷100). Default value is 100.
INP[n]	Boolean operand: it indicates nth digital input.
OUT[n]	Boolean operand: it indicates nth digital output.
Q[index]	Indexed position parameter. 50 floating point parameters are fixed

	(indexes between 0÷49) in order to define quotes in engineering units.
F[index]	Indexed flag. 50 Boolean flags are fixed (indexes between 0÷49) for static logic memorization.
T[index]	Indexed timing parameter. 50 floating point parameters are fixed (indexes between 0÷49) to define times in seconds.
P[index]	Indexed general-purpose parameter. 50 integer parameters are fixed (indexes between 0÷49) in order to manage values in the application.
ON	Boolean constant used for assignment and comparison between Boolean operand.
OFF	Boolean constant used for assignment and comparison between Boolean operands.
constant	It is possible to use both integer and floating point numeric constant for assignment and comparison.
[name]:	Definition of a marker (label) inside the program.
START	Implicit label, meaning the start of program.
END	Implicit label, meaning the end of program.

Operators

a := b	Assignment operator. It permits to assign the value of b operand to a operand.
a = b	Comparison operator: it carries out comparison between a e b operands. The Boolean result is used to execute conditional jumps.
NOT a	Boolean negation operator. It makes negation of a operand.
WAIT c	Waiting operator. It stops program execution till c condition gets true.
INCR a	Increment operator. It adds unitary increment to a operand. General-purpose parameters are the only ones allowed for this operator.
DECR a	Decrement operator. It subtracts unitary decrement from a operand. General-purpose parameters are the only ones allowed for this operator.
; comm.	Explanatory note operator. It means that the elements, which follow till the end of the line, must be considered as program comment.

Commands

GOTO p	It branches to p position: p is a line number or a label.
STOP	It stops the program. The program keeps active itself and it restarts the execution from the next instruction by a start command. If a positioning is in progress, it is stopped with programmed deceleration.
M [index]	It starts the execution of a PLC macro-command. It is possible to activate more than one macro at the same time, also if in concomitance of a positioning command. Simultaneous activation is obtained by writing macro commands in the same line The maximum number of macro commands is 64.

Constructs

--	--

IF c ... ELSE ... ENDIF	<p>When c condition is true, program lines between IF and ELSE instructions are executed: otherwise, program lines between ELSE and ENDIF instructions are executed.</p> <p>ELSE instruction is optional: so the construct can become IF ... ENDIF.</p> <p>This construct is not nesting, that is to say, it is not possible to insert an IF ... ELSE ... ENDIF sequence just inside another sequence of the same type.</p>
WHEN ... ENDWHEN	<p>Program lines between WHEN and ENDWHEN instructions are executed at the same time of last fixed positioning.</p> <p>Inside this construct, it is possible to make comparisons between actual position and fixed values.</p> <p>It is possible to combine a set of instruction to each quote, which is executed when the quote itself has been reached.</p> <p>This construct is not nesting, that is, it is not possible to insert a WHEN and ENDWHEN sequence just inside another sequence of the same type.</p>

Positioning execution

The positioning is obtained by establishing a fixed value into positioning register:

X := quote	It carries out a positioning. "quote" is a floating point constant expressing wanted position in fixed unity of measurement.
X := Q[index]	It carries out a positioning to the quote corresponding to the value of the indicated quote parameter.
X += quote	The same as X := quote but the quote is interpreted as positive incremental movement value.
X -= quote	The same as X := quote but the quote is interpreted as negative incremental movement value.
X += Q[index]	The same as X := Q[index] but the Q parameter is interpreted as positive incremental movement value.
X -= Q[index]	The same as X := Q[index] but the Q parameter is interpreted as negative incremental movement value.
SPD := value	It sets the value of the speed to use during positioning. It expresses the percentage of maximum speed in the corresponding system parameter. Value equal to 100 is equivalent to the maximum speed. Values less or equal to 0 are not admitted.
ACC := value	It sets the value of the acceleration to use during positioning. It expresses the percentage of maximum acceleration in the corresponding system parameter. Value equal to 100 is equivalent to the maximum acceleration. Values less or equal to 0 are not admitted.

Assignments

In addition to positioning execution, assignment instructions can be used to operate both on digital output status and values of parameters. In the follow section, we explain other possible

assignments:

OUT[n] := ON	It gets n th digital output active.
OUT[n] := OFF	It gets n th digital output not active.
OUT[n] := INP[m]	It copies the status of indexed m digital input on indexed n digital output.
P[n] := value	It assigns the fixed value to indexed n general-purpose parameter.
T[n] := T[m]	It assigns the value of indexed m time variable to indexed n time variable.
T[n] := valore	It assigns the fixed value to indexed n time variable.
Q[n] := Q[m]	It assigns the value of indexed m quote variable to indexed n quote variable.
Q[n] := valore	It assigns the fixed value to indexed n quote variable.
F[n] := valore	It assigns the fixed value to indexed n flag variable. (only boolean value ON - OFF)
F[n] := F[m]	It assigns the value of indexed m flag variable to indexed n flag variable.
F[n] := INP[m]	It copies the status of indexed m digital input into indexed n flag variable.

Wait execution

WAIT instruction permits to stop program execution, waiting for a particular status of digital input or flags, or expiring a timeout.

WAIT INP[n] = OFF	It waits for the indexed n digital input got not active.
WAIT F[n] = ON	It waits for the indexed n digital input got active.
WAIT	It waits for a timeout in seconds.
WAIT T[index]	It waits for a timeout fixed by indexed time parameter.

Jump execution

AML language gives you the possibility of making conditional and unconditional jumps. Conditional jumps are made by using IF [condition] .. ELSE .. ENDIF.

In the following section, we explain some conditions used inside the IF construct:

IF INP[n] = ON	It performs the instructions of IF block, if indexed n input is active.
IF P[n] = P[m]	It performs the instructions of IF block, if indexed n parameter is equal to indexed m parameter.
IF INP[n] = F[m]	It performs the instructions of IF block, if indexed n input is equal to indexed m flag.

Unconditional or absolute jumps, are made by mean of GOTO instruction, using the following forms:

GOTO label	It makes an unconditional jump to program instruction indicated by "label".
------------	---

Operations during positioning

WHEN .. ENDWHEN construct allows indicating operational sequences, which have to be executed when, during a positioning, the reached quote is equal to a fixed target used inside the construct. The form is:

X := position	Positioning instruction.
WHEN	Starting operations to execute during positioning.
X = trag 1	Set 1 st target.
..	Operations to execute when 1° target is reached.
X = trag n	Set nth target
..	Operations to execute when nth target is reached.
ENDWHEN	

Contemporaneous commands

AML language gives you the possibility to execute contemporaneous commands, like positioning and Macro commands.

To execute at the same time these commands, they have to be written in the same line. Their syntax is:

X := position M20 M21	Positioning instruction with activation of two macro commands at the same time.
-----------------------	---

APPLICATION NOTES

The present section presents some programming hints and application notes.

- [The AML START default application](#)
- [Circular axes](#)
- [Macro programming](#)
- [PLC interface example](#)

The AML START default application

Few notes about AML START application.

- This application is the default PLC application distributed by Phase Motion Control to perform basic AML setup.
- It's a generic application but it can be applied in a lot of application cases.
- The homing cycle included is configurable and allows to perform homing with sensor+encoder index, only index, only sensor and also only on drive enable.
- The digital inputs can be configured in order to have separate start/stop commands, jog commands, single-step function and so on.
- Macro routines are obviously not included, but they can be added easily by editing the AMLSLOW.PLC module of the PLC project (macros can be put at the end of the program)
- The application is also a complete example on how to realize a complete PLC base for the AML controller; therefore it can be "saved as.." in order to realize new applications.

Circular axes

Circular axes behavior (like rotating tables) can be easily obtained with AML environment with these simple steps:

- Put the same value into the [QMIN](#) and [QMAX](#) system parameters (for example 360, for the degrees into a revolution).
- Use the positive or negative [incremental movements](#) to make steps into clockwise or counterclockwise direction.
- Use [absolute movements](#) to execute positioning at an absolute angular position.

Macro programming

The following is an example on how to perform a simple macro interfacing with AML and PLC. In this example we suppose that macro n° 2 executes a spindle rotation for 2 secs. This is a very simple example but illustrates the potentialities that can be reached with mixed programming.

- AML program:

```
.
.
X := 100      ; Goto position 100
M2           ; Execute macro n° 2 (we suppose that this command runs a
             ; spindle for 2 secs)
.
.
```

- PLC program:

```
.
.
LD          amlM2      ; Test in Macro 2 is activated by AML
JMPCN      lEndM2     ; and jump away if not active
LD          startMacro ; Check if first PLC cycle of macro execution
JMPCN      lRunM2     ; if not, jump to normal macro execution
LD          0
ST          cycleCount ; Reset the macro time counter
LD          FALSE
ST          startMacro ; Reset the start flag
```

lRunM2:

```
LD          2048
ST          aout0      ; Puts +10V on analog output 0 to run the spindle
LD          cycleCount ; Increment time counter
ADD         1
ST          cycleCount
LD          cycleCount
EQ          250       ; Test if 2 secs elapsed (assuming 8 ms PLC
                    ; cycle)
JMPCN      lEndM2     ; if not jump to the rest of PLC execution
LD          0
ST          aout0      ; Puts 0V to stop the spindle
```

```

LD      FALSE
ST      amlM2      ; Inform AML that the macro is terminated,
                ; AML will execute the rest of the program
LD      TRUE      ; Reload start flag for next macro execution
ST      startMacro

```

```
lEndM2:
```

```

.
.

```

PLC interface example

The following example shows another example of AML expansion with the use of PLC. We suppose that Q1 and Q2 are a target position and a position increment respectively. The amount of increment depends on an external encoder position (a typical stacking application).

The AML program look like this:

```

.
.
Q1 := 100      ; Base position
WAIT F1 = ON   ; Wait for Q1 calculation completed
X := Q1        ; Goto the calculated position Q1
.
.

```

The PLC program can be the following:

```

.
.
LD      Di_ViPo      ; Read the external encoder
TO_REAL
MUL     kInp2Uu      ; Apply the conversion factor between user units
                ; the encoder counts
ST      amlQ2        ; Store the value into AML Q2 (just for
monitoring)

LD      amlQ2        ; Add the position increment to Q1
ADD     amlQ1
ST      amlQ1

LD      TRUE         ; Inform AML that the new position is ready
ST      amlF1
.
.

```