PLCopen

*Standardization in Industrial Control Programming*

# Creating Reusable, Hardware Independent Motion Control Applications via IEC 61131-3 and PLCopen Function Blocks for Motion Control

Eelco van der Wal, PLCopen, Zaltbommel, The Netherlands, evdwal@plcopen.org

***Abstract***

Motion integration issues have emerged to the forefront, along with maintainability and connectivity to automation solutions. For this standards are needed.

PLCopen has generated such a standard. A standard in programming language, to harmonize the access of motion control functionality across platforms. In this way, the generated application program is much more hardware independent, and re-usable across platforms.

The provided standard, the PLCopen Function Blocks for Motion Control, is based on IEC 61131-3 Function Block concept. With the standardization of the functionality and the interfaces, as well as implementations on multiple platforms, it provides a programming standard that is widely supported by the industry.

Due to the data hiding and encapsulation, it is usable on different architectures, for instance ranging from centralized to distributed or integrated to networked control. It is not specifically designed for one application, but will serve as a basic layer for ongoing definitions in different areas. As such it is open to existing and future technologies.

## I. INTRODUCTION

Users of motion control systems in many cases support multiple application levels. For instance a supplier of packaging machines can support three different levels: a low cost, a medium level and a high-performance / high costs machine. The different requirements in speed and accuracy will be solved by using different motion control systems. Unfortunately, the motion control market shows a wide variety of incompatible systems / solutions. In practice this means that the architecture and the software tools for development, installation and maintenance will differ widely per level.

This incompatibility induces considerable costs: applying different implementations is confusing, engineering becomes difficult, training costs increase, and the software is not reusable across platforms.

Standardization certainly reduces the negative factors above. Standardization not only in the programming languages itself, like done within the worldwide IEC 61131-3 standard, but also the interface towards different motion control solutions, like distributed, integrated, or centralized.

Effectively, this standardization is done by defining libraries of reusable components. In this way the programming is less hardware dependent, the reusability of the application software increased, the cost involved in training and support reduced, and the application becomes scalable across different control solutions.

## II. GOALS OF THE TASK FORCE

Leading suppliers from the machine building industry have requested PLCopen to help to solve this incompatibility problem, which initiated the Task Force Motion Control. This Task Force defined the programmer's interface by standardizing the Function Blocks for Motion Control. Not only users are involved: with wide support from the suppliers, providing numerous implementations across a broad range of products and architectures, success of this library is guaranteed.

The result, a definition of a library of Function Blocks, is published by the independent association PLCopen in November 2001 on its website www.plcopen.org . It will be maintained (and implemented) by the members of this association. Compliance rules are included in this definition, and will be certified by PLCopen.

The standardization is primarily focused on reusability across different systems of different suppliers, including distributed and networked systems. Overall, the standardization is expected to cover around 80% of the motion control market.

The task force has defined the following goals for the definition of the motion control function blocks:

1. Simplicity - ease of use, towards the application program builder and installation & maintenance
2. Efficiency - in the number of blocks, directed to efficiency in design (and understanding)
3. Consistency - conforming to IEC 61131-3 standard
4. Universality - hardware independent
5. Flexibility - future extensions / range of application
6. Completeness - not mandatory but sufficiently

The definition contains three parts:

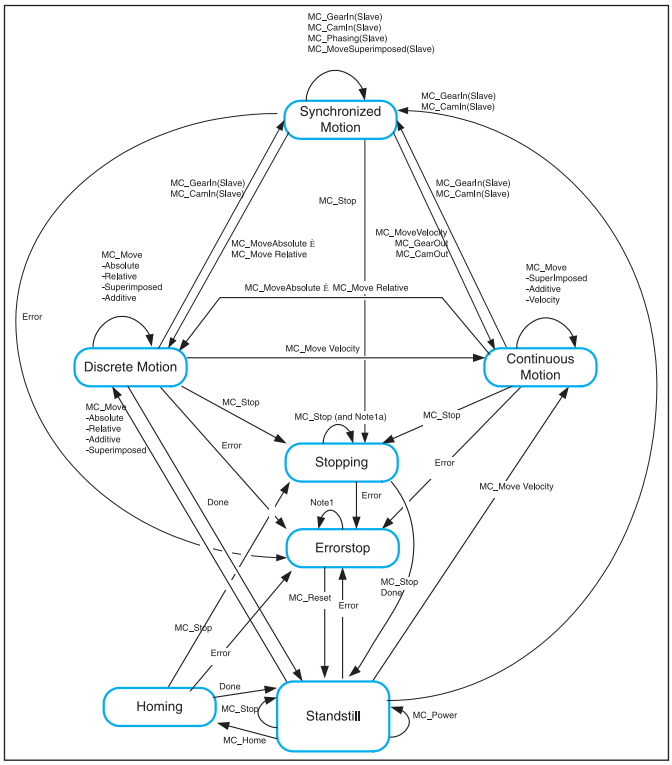
1. definition of the state machine
2. definition of a basic set of FB's for single axis multi-axes motion control
3. Compliance rules and statement

## III. Definition of the state machine

The following diagram normatively defines the behavior of the axis at a high level when multiple motion control FB are «simultaneously» activated. This combination of motions is useful to build a more complicated profile or to treat exceptions within a program. (In real implementations there may be additional states at a lower level defined).

The basic rule is that motion commands are always taken sequentially, even if the PLC had the capability of real parallel processing. These commands act on the axis' state diagram.

The axis is always in one of the defined state (see diagram). Any motion command is a transition that changes the state of the axis and, as a consequence, modifies the way the current motion is computed.




There are seven states defined:

1. Stand Still
2. Homing
3. Discrete Motion
4. Continuous Motion
5. Synchronized Motion
6. Stopping
7. Error Stop

A normal procedure would start in Stand Still. In this state the power can be switched on per axis (via the command Power, see above). Also, one can access the Homing state (via the issue of the command Home per axis), which after normal completion returns to Stand Still. From here one can transfer an axis to either Discrete Motion or Continuous Motion. Via the state Stopping one can return to StandStill. ErrorStop is a state the axis transfers to in case of an error. Via a (manual) Reset command one can return to StandStill, from which the machine can be moved to an operational state again. Please note that the States define the functionality of the Function Blocks.

## IV. DEFINITIONS FOR THE SET OF FUNCTION BLOCKS

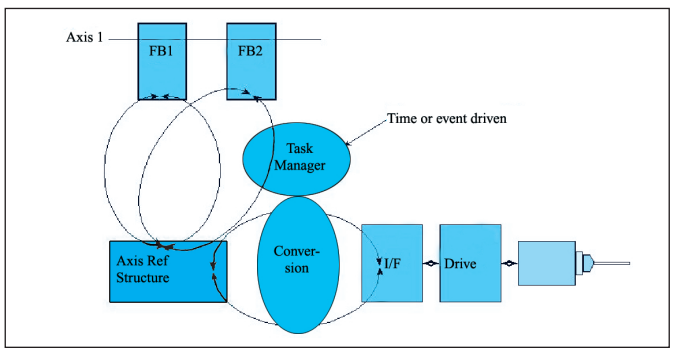
A basic problem concerns the granularity of the function blocks. The extremes are one function block per axis versus a low, command level functionality. An in-between level fulfils better the guidelines above, saves processor load and creates a higher level of scalability. This is the level where the work is based upon.
In addition, for future extensions, two sets for the inputs and outputs of the Function Blocks are defined: a mandatory Basic Set and an Extended set.

A. Axis Datatype & instantiation

The reference to an axis is done via the derived datatype AXIS_REF. This datatype is supplied by all manufacturers . It provides the interface towards the motor / drive itself. The technicalities of the real interface towards are hidden within the structure and function block itself. In this way a different architectures looks the same to the user while giving access to all relevant parameters. As such it covers architectures like motion integrated in the controller, control integrated in the drive, distributed and networked systems.

*B. AxisRef as Var_In_Out*

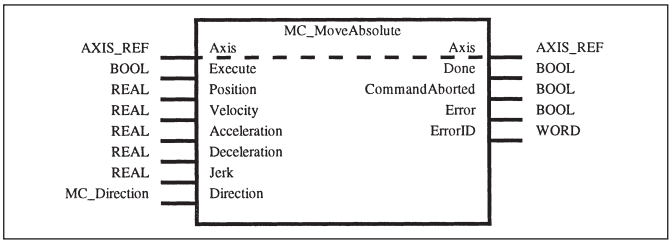
The Axis_Ref is used as Var_In_Out, represented as an input and an output connected by a horizontal line in a graphical representation of a Function Block. The value of an input/output variable is stored externally to the FB. The variables used within Axis_Ref, acting both as input and output parameters, can be modified within the Function Block, as well as receive values from external variables.
As an example of how this could operate: imagine a Program containing several function blocks, all linked after each other (left-to-right) and all referring to the same axis via Axis_Ref (see below). Also, this Program is in a cyclic task-mode, for instance every millisecond. The first FB reads the latest values in Axes_Ref, and might update some of these values before it finishes its execution. Then the next FB is started and reads the updated values within Axes_Ref, so uses the latest values. And these values are coupled to the motor itself. Again, the control architecture can be quite different across systems.
One can use this reference to define one or more virtual axes, in that sense that it exists as a datastructure but is not coupled to a physical drive and/or motor.




## V. The Defined function blocks for single axis motion control

| FB-Name | **MC_MoveAbsolute** |
|---|---|
| This function block commands a controlled motion at a specified absolute position. | |

Representation:



The other single axis Function Blocks are listed here below in shortform:

**MC_MoveRelative**
commands a controlled motion of a specified distance relative to the actual position at the time of the execution.

**MC_MoveAdditive**
commands a controlled motion of a specified relative distance additional to the original commanded position in the discrete motion state. If the FB is activated in the Continuous Mode the specified relative distance is added to the actual position at the time of the execution.

**MC_MoveSuperimposed**
commands a controlled motion of a specified relative distance additional to an existing motion. The existing Motion is not interrupted, but is superimposed by the additional motion.

**MC_MoveVelocity**
commands a never ending controlled motion at a specified velocity.

**MC_Home**
commands the axis to perform the «search home» sequence. The details of this sequence are manufacturer dependent and can be set by axis' parameters. The position input is used to set the absolute position when reference signal is detected. It completes at standstill.

**MC_Stop**
commands a controlled motion stop and transfers the axis to the state "Stopping". It aborts any ongoing function block execution. With the Done output set, the state is transferred to the StandStill. While the axis is in state Stopping, no other FB can perform any motion on the same axis.

**MC_Power**
controls the power stage (on or off).

**MC_ReadStatus**
returns in detail the status of the axis with respect to the motion currently in progress.

**MC_ReadAxisError**
indicates errors not relating to the function blocks.

**MC_Reset**
makes the transition from the state ErrorStop to StandStill by resetting all internal axis-related errors and clearing pending commands – it does not effect the output of the FB instances.

**MC_ReadParameter & MC_ReadBoolParameter**
returns the value of a vendor specific parameter. The returned Value has to be converted to Real if necessary. If not possible, the vendor has to supply a supplier dependent FB for it.

**MC_WriteParameter & MC_WriteBoolParameter**
modifies the value of a vendor specific parameter.

**MC_ReadActualPosition**
returns the actual position.

**MC_PositionProfile**
commands a time-position locked motion profile

**MC_VelocityProfile**
commands a time-velocity locked motion profile

**MC_AccelerationProfile**
commands a time-acceleration locked motion profile

## VI. Common set of multi-axes Function Blocks

For multi-axes, coordinated movements, a small set is defined. This set will be extended by additional application specific libraries. The current defined Function Blocks are:

**CamTableSelect**
selects the CAM tables by setting the pointers to the relevant tables.

**CamIn**
engages the CAM.

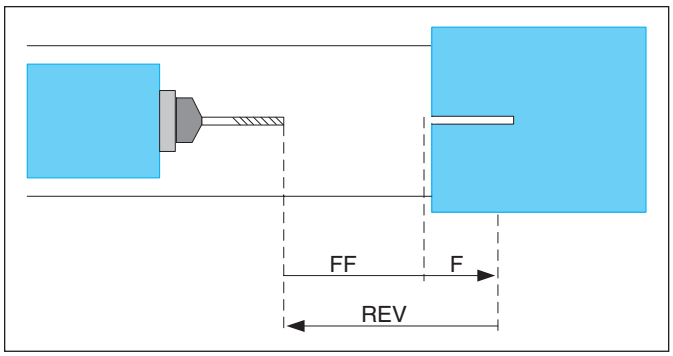
**CamOut**
disengages the Slave from the Master axis iimediately.

**GearIn**
commands a ratio between the VELOCITY of the slave and master axis.

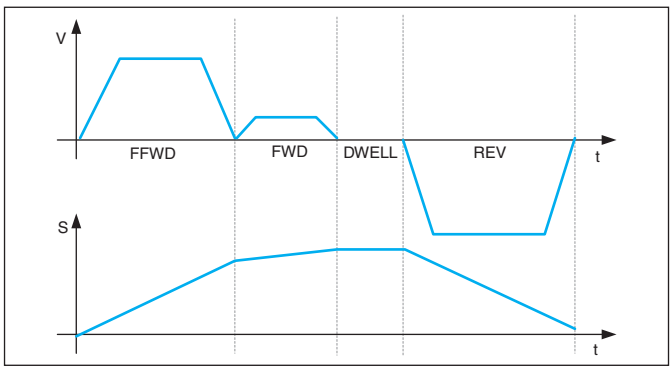
**GearOut**
disengages the Slave from the Master axis

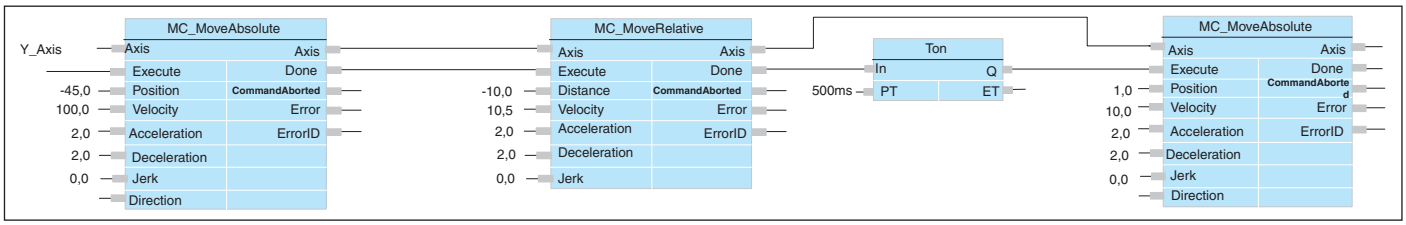
## VII. An example

The following example is Example of a simple drilling unit:
Example of a simple drilling unit




The corresponding timing diagram for the movement is as follows:





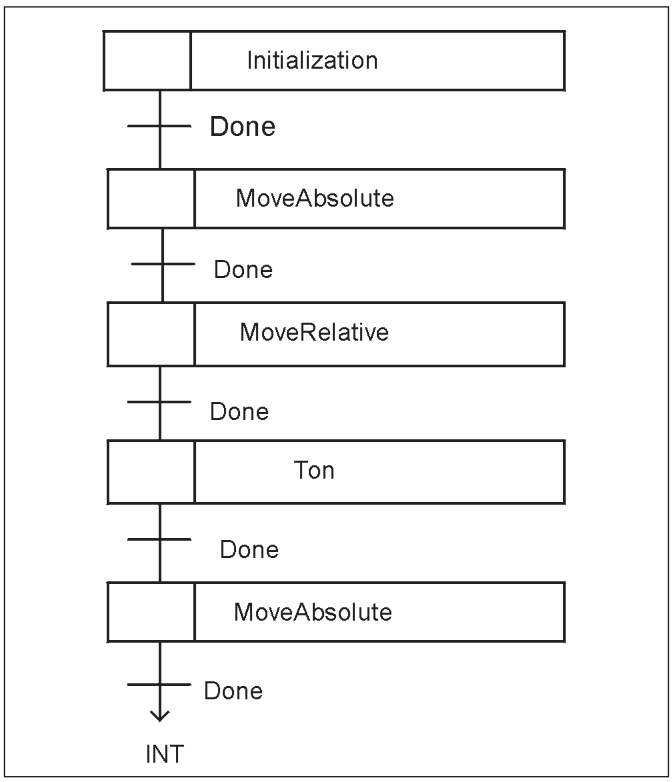

*A. Solution with Function Block diagram*

*A. Sequential Function Chart*

This is the more traditional approach using Sequential Function Charts for the specification of sequencing steps.

The SFC reassembles the timing diagram given in the example above. It consists of the following steps:

*Step 1:* Initialization, for instance at power up.




*Step 2:* Move forward to drilling position and start driller turning: in this way it will be fully operational before the position is reached; then check if both actions are completed.

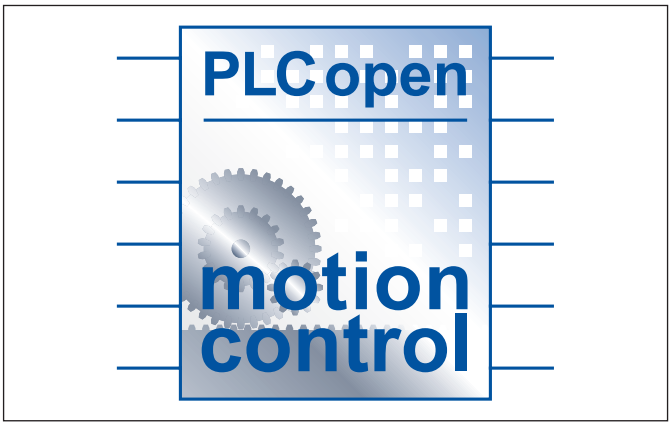
*Step 3:* Drill the hole.

*Step 4:* After Drilling the hole we have to wait for the step-chain sequence to finish dwelling the hole free of any stuff which might have stuck in the hole.

*Step 5:* Move driller back to starting position and shut the spindle off. Combining the finishing of moving backwards and stopping the spindle we signal the step-chain to start over.

## VIII. CERTIFICATION

Included in the document are the rules for compliance and certification. Basically this is a self-certification, from which the results per supplier are published on the PLCopen website www.plcopen.org . Certified companies are allowed to use the logo below, with additional number, date and number of supported compliant function blocks:




## IX. CURRENT RESULTS OF THE TASK FORCE

The document has been released in November 2001. It contains all elements as described above. In addition, the first implementations are done, which resulted in feedback which increased the quality of this standard. Investigations into the real-time behavior is done, and its mapping onto the IEC 61131-3 environment. Due to its multi-tasking nature and time synchronization aspects, implementations put additional requirements on the target hardware.

With a broad range of motion applications to be covered, we realize that this first release will not be the end. Further libraries for specific applications areas will be created on top of this basic specification. The first area is for packaging applications, adding for instance Linear and Circular Interpolation and Blending to the multi-axes set.

## X. CONCLUSION

The finalization of release 1 and the first implementations clearly shows that the multi-axes implementation certainly fits within the framework as defined by the IEC 61131-3 standard. This means that the standard itself does not inhibit this set of function blocks for motion control, although certain implementations of this standard are expected to be less suitable for these complex tasks.

The first implementations of this set are currently available. After this, motion control will never be the same: more hardware independence give the users less training costs, and the possibility to create application software that is usable on a number of different targets more easily, and which can be selected in a late phase of the overall machine development cycle. This brings a higher efficiency and reduces costs during development, maintenance and training.

For up-to-date information as well as the specification itself, check www.plcopen.org.