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

I. INTRODUCTION

Based on application requirements and project specifications engineers are required to use a wide range of motion control hardware. In the past this required unique software to be created for each application even though the functions are the same. The PLCopen motion standard provides a way to have standard application libraries that are reusable for multiple hardware platforms. This lowers development, maintenance, and support costs while eliminating confusion. In addition, engineering becomes easier, training costs decrease, and the software is reusable. 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. Due to the data hiding and encapsulation, it is usable on different architectures, ranging from centralized to distributed or integrated to networked control. It is not specifically designed for one application, but will serve as a basis for ongoing definitions in different areas. As such it is open to existing and future technologies.

II. OVERVIEW OF THE RESULTS

Currently the suite of PLCopen Motion Control Specifications consists of the following parts:

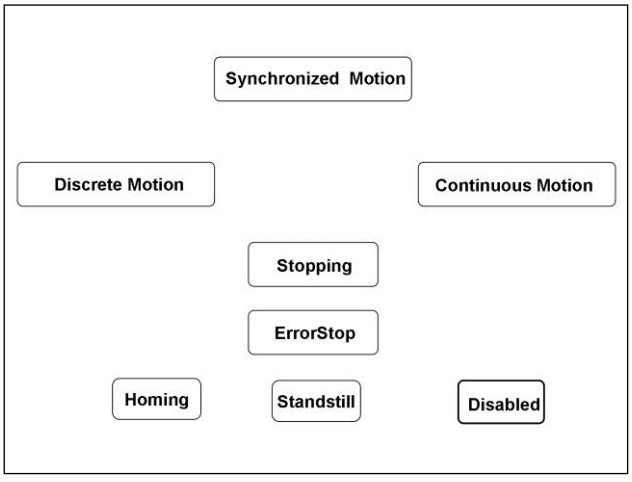
- Part 1& 2 – Basics & Extensions combined in one
- Part 3 – User Guidelines and examples
- Part 4 – Coordinated motion
- Part 5 – Homing Procedures
- Part 6 – Extensions for fluid power

Basically every specification contains 3 sections:

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

III. DEFINITION OF THE STATE MACHINE

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.
The state diagram normatively defines the behavior of the axis at a high level. This diagram 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).
There are eight states defined as shown in the picture below:



A normal procedure starts in Disabled. In this state the power can be switched on per axis (via the command MC_Power) which transfers the relevant axis to the state Standstill. From there 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. From these states a coupling to a master axis can be realized for instance via issuing MC_GearIn. The resulting state for the slave axis is then Synchronized Motion. Issuing a single axis move command will bring the axis back to either Discrete 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.

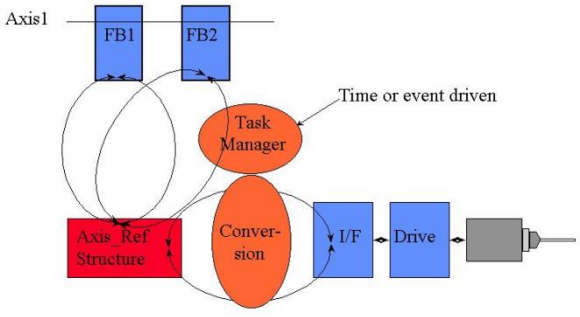
IV. FUNCTION BLOCKS DEFINITIONS

A. *AxisRef*

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 are hidden within the structure and function block itself. In this way, different architectures, from centralized to distributed and networked systems, looks the same to the user while giving access to all relevant parameters.

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 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. However they are stored externally to the FB, making copying of the structure unnecessary.
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. 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 internally 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.

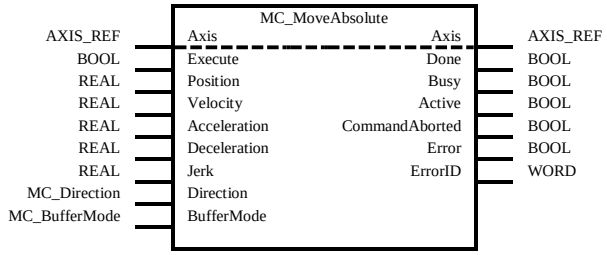


V. THE FUNCTION BLOCKS FOR SINGLE AXIS MOTION CONTROL

There are motion related and administrative function blocks defined in both part 1 and 2. The first movement function block is shown here in some more detail:

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

In a graphical representation it looks like:



The other single axis Function Blocks as defined in part 1 and 2 are listed here below in short-form:

- **MC_MoveRelative** - moves the axis a specified distance relative to the actual position at the time of the execution;
- **MC_MoveAdditive** - for a specified relative distance additional to the original commanded position in the discrete motion state. In state Continuous Motion the specified relative distance is added to the actual position at the time of the execution;
- **MC_MoveSuperimposed** - for a specified relative distance additional to an existing motion. The existing Motion is not interrupted, but is superimposed by the additional motion;
- **MC_HaltSuperimposed** - commands a halt to all superimposed motions of the axis. The underlying motion is not interrupted;
- **MC_MoveVelocity** - for a never ending controlled motion at a specified velocity;
- **MC_MoveContinuousAbsolute** &**MC_MoveContinuousRelative** - commands a controlled motion to a specified absolute or relative position ending with the specified velocity;
- **MC_TorqueControl** - continuously exerts a torque or force of the specified magnitude, approached using a defined ramp, and sets the 'InTorque' output if the torque level is reached;
- **MC_SetPosition** - shifts the coordinate system of an axis by manipulating both the set-point position as well as the actual position of an axis with the same value without any movement caused;
- **MC_SetOverride** - sets the values of override for the whole axis and all functions that are working on that axis;
- **MC_TouchProbe** - records an axis position at a trigger event;
- **MC_AbortTrigger** - is used to abort function blocks which are connected to trigger events (e.g. MC_TouchProbe);
- **MC_DigitalCamSwitch** - provides the analogy to switches on a motor shaft: it commands a group of discrete output bits to switch in analogy to a set of mechanical cam controlled switches connected to an axis. Forward and backward movements are allowed;
- **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, as well as the function blocks as defined in Part 5 – Homing Procedures;
- **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_Halt** - commands a controlled motion stop. It aborts any ongoing function block execution. The axis is moved to the state 'DiscreteMotion' until the velocity is zero. With the 'Done' output set, the state is transferred to 'StandStill';
- **MC_Power** - switches the power stage on or off;
- **MC_ReadStatus** - returns in detail the status of the state diagram of the selected axis;
- **MC_ReadMotionState** - returns in detail the status of the axis with respect to the motion currently in progress;
- **MC_ReadAxisInfo** - reads information like modes, inputs directly related to the axis, and certain axis status information;
- **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;
- **MC_ReadParameter** & **MC_ReadBoolParameter** - Returns the value of a vendor specific parameter;
- **MC_WriteParameter** & **MC_WriteBoolParameter** - modifies the value of a vendor specific parameter;
- **MC_ReadActualPosition** - returns the actual position;
- **MC_ReadDigitalInput** - provides the value of the digital input as referenced by INPUT_REF;
- **MC_ReadDigitalOutput** - provides the value of the digital output as referenced by OUTPUT_REF;
- **MC_WriteDigitalOutput** - writes a value to the output referenced by the argument 'Output' once;
- **MC_ReadActualVelocity& MC_ReadActualTorque** - returns the value of the actual velocity or torque as long as enabled;
- **MC_PositionProfile, MC_VelocityProfile** & **MC_Acceleration-Profile** - commands different locked motion profiles: a time-position, a time-velocity or a time-acceleration 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 immediately;
- **GearIn** - commands a ratio between the velocity of the slave and master axis;
- **GearOut** - disengages the Slave from the Master axis;
- **MC_GearInPos** - commands a gear ratio between the position of the slave and master axes from the synchronization point onwards;
- **MC_PhasingAbsolute** & **MC_Phasing Relative** - creates an absolute or relative phase shift in the master position of a slave axis;
- **MC_CombineAxes** - combines the motion of two axes into a third axis with selectable combination method.

A. Aborting, merging, and blending

Multiple function blocks have an input to set the different operating modes, combined with an output for signalling this. With this input, the FB can either work in a "Non-buffered mode" (default behavior) or in a "Buffered mode". The difference between those modes is when they should start their action:

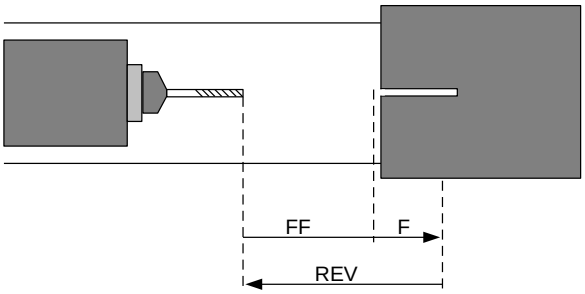
- A command in a non-buffered mode acts immediately, even if this interrupts another motion;
- A command in a buffered mode waits till the current FB is done (signaled via the corresponding output or via in-position, or in-velocity or similar outputs).

The following modes have been identified:

- *Aborting* - Default mode without buffering. The next FB aborts an ongoing motion and the command is affecting the axis immediately;
- *Buffered* - The next FB is affecting the axis as soon as the previous movement is 'Done'. There is no blending;
- *BlendingLow* - The next FB is controlling the axis after the previous FB has finished (equal to buffered), but the axis may not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1);
- *BlendingPrevious* - blending with the velocity of FB 1 at end-position of FB 1;
- *BlendingNext* - blending with velocity of FB 2 at end-position of FB1;
- *BlendingHigh* - blending with highest velocity of FB 1 and FB 2 at end-position of FB1.

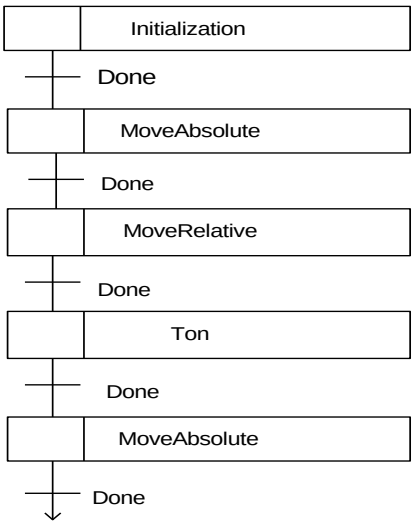
VII. An example

The following example is Example of a simple drilling unit:



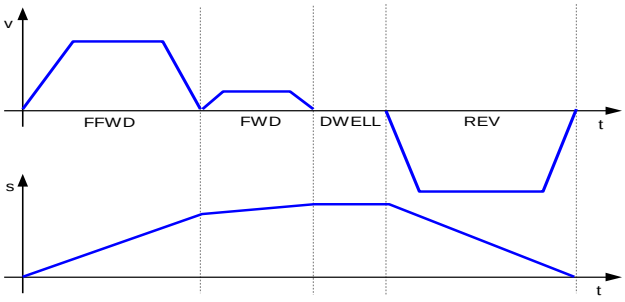
We use Sequential Function Chart here to describe the different step for this drilling example.

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.

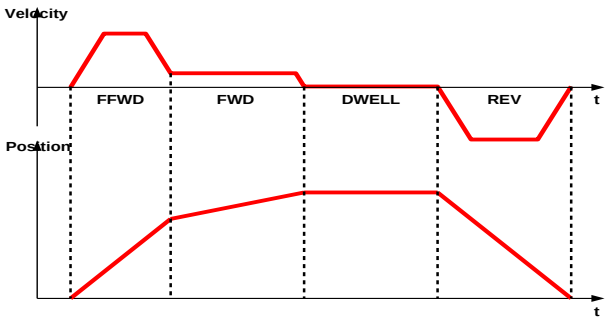


Representation of the drilling example in SFC

The corresponding timing diagram for the movement is depending on the selected mode. For example:



Timing diagram for drilling – aborting mode



Timing diagram for drilling – blending mode

VIII. Part 3 – PLCopen Motion Control User Guidelines

Within Part 3 user examples are explained. These examples show how to create real application parts based on the function blocks as defined in the specifications. With this, a user can create an own set of Function Blocks, and so their own library, dedicated to their typical application areas.
To give an example: in Part 2 there is no registration function defined. For this the function block TouchProbe is intended. With an example it is shown how the user can create an own function block Registration, based on the function block TouchProbe, together with already defined function blocks. With this, an application specific function block is created that also can be used across platforms. In this way, different inputs from sensors can be used, where either the location is not known (directly or networked) and /or need to be compensated. This user derived function block can be added to the company own library. This registration function now can be used company wide, and the source is usable on different platforms. This saves time and money in the next machine.
This part is an ongoing specification, which releases as new examples are added. For up-to-date information, check www.PLCopen.org.

IX. Part 4 – Coordinated Motion

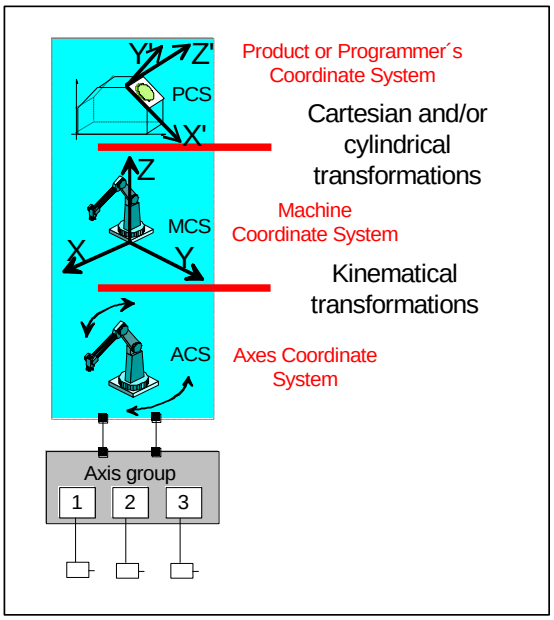
Part 4 is focused to the coordinated multi-axes motion in 3D space, to serve the majority of user's application needs in this area. Part 1 and Part 2 deal with Master / slave motion control, a type of coordinated motion control where the master axis position is used to generate one or more slave axis position commands. For multi-dimensional movements, one goes beyond this point via a grouping of a set of axes, without a master axis. This is done via the definition of a set of Function Blocks with related coordinated motion functionality as well as a higher level state diagram, linking the single axis state diagrams in the group. In this way a better trajectory planning is possible. Also, the current Master/Slave axes can have the problem that if an error occurs, the other axes have no knowledge about this, and continue their movement. By combining axes in a group one knows upfront which axes are involved and has the basis for a better error behavior.
The level of the PLCopen Motion Control Function Blocks are specified at such a level that the user quickly recognizes the functionality of the function block and what happens if it is activated or connected to other blocks in a sequence of motion commands. This PLCopen initiative transforms the functionalities as known in the CNC and Robotic world to the PLC world.
Overall there are a number of sets of function blocks defined to do this, as shown below without going into details: for this download the specification from the website www.PLCopen.org.

The first step is to group axes. The following FBs are defined for this:

- MC_AddAxisToGroup
- MC_RemoveAxisFromGroup
- MC_UngroupAllAxes
- MC_GroupReadConfiguration
- MC_GroupEnable
- MC_GroupDisable

The next step is to link the transformations:

- MC_SetKinTransform
- MC_SetCartesianTransform
- MC_SetCoordinateTransform
- MC_ReadKinTransform
- MC_ReadCartesianTransform
- MC_ReadCoordinateTransform
- MC_SetDynCoordTransform



Overview of the coordinate systems and transformations

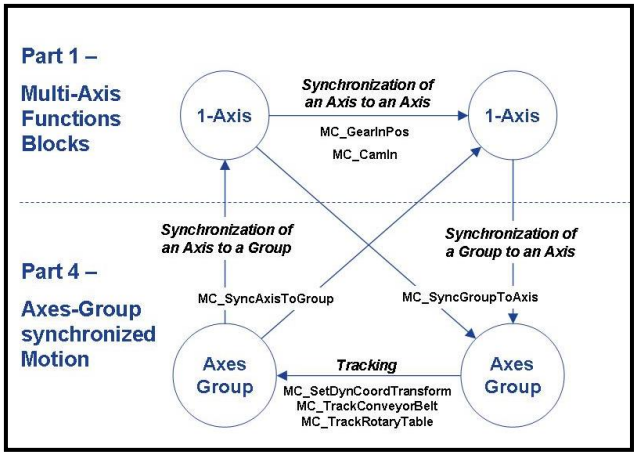
- ACS Axis related
- MCS Machine related
- PCS Product or Workpiece related

For coordinated moving, the following blocks are defined:

- MC_GroupHome
- MC_GroupStop
- MC_GroupHalt
- MC_GroupInterrupt
- MC_GroupContinue
- MC_MoveLinearAbsolute
- MC_MoveLinearRelative
- MC_MoveCircularAbsolute
- MC_MoveCircularRelative
- MC_MoveDirectAbsolute
- MC_MoveDirectRelative
- MC_MovePath

Axes Group Synchronized Motion

The following function blocks deal with a master/ slave relationship between a single or group of axes and a single or group of axes for coordination purposes. There are two kinds of coordinated motion that have to be distinguished from a programming point of view and in the realization of the motion control itself. These two modes are identified here through their names: Synchronization and Tracking. The differences and related FBs are shown in the next figure:



Graphical explanation of coordination

Then there are other FBs defined, most without motion:

- MC_GroupSetPosition
- MC_GroupReadActualPosition
- MC_GroupReadActualVelocity
- MC_GroupReadActualAcceleration
- MC_GroupReadStatus
- MC_GroupReadError
- MC_GroupReset
- MC_PathSelect
- MC_GroupSetOverride

X. Part 5 – Homing Procedures and Blocks

A. *Homing Procedures*

The homing functionality was originally seen as a separate sequence in the startup phase of a machine or axis. The sequence for an axis was: power on, homing, and move 'something'. The whole homing procedure itself was hidden to the user. However, some users needed more control on the homing functionality itself. For this reason, a set of building blocks have been identified to be able to define dedicated homing procedures. This procedure can be encapsulated in a dedicated homing function block, which can be added to the library and used for this specific procedure. As examples, several possible homing procedures are defined:

- *HomeAbsSwitch* - Absolute Switch homing plus limit switches;
- *HomeLimitSwitch* - Homing against limit switches;
- *HomeBlock* - Homing against hardware parts blocking movement;
- *HomeRefPulse* - Homing using encoder reference pulse 'Zero Mark';
- *HomeRefPulseSet* - Homing using a set of encoder reference pulses 'Zero Mark';
- *HomeDirect* - Static Homing, forcing a position from a user reference;
- *HomeAbsolute* - Static Homing, forcing a position from an absolute encoder.

B. *Homing Step Function Blocks*

To make these procedures available to the user, a toolkit is defined with additional Function Blocks, FBs. By using one or more connected function blocks one can create complex homing sequences. With these, a user can create its own homing procedure in more detail, and even create own, user specific, homing procedures. Additionally, these function blocks include advanced homing process error reporting, evaluating time, distance and torque limits.

The following defined FBs match the homing procedures and are issued in or change the state to the 'Homing' state.

- **MC_StepAbsSwitch** - Absolute Switch homing plus limit switches;
- **MC_StepLimitSwitch** - Homing against limit switches;
- **MC_StepBlock** - Homing against hardware parts blocking movement;
- **MC_StepRefPulse** - Homing using encoder reference Pulse 'Zero Mark'.

In this way, any combination sequence is possible without the need of predefining hundreds of homing methods. The individual adjustments for torque limit, time limit and distance limit provide control of the sequence error conditions.

The following FBs lead to a final position and leave the homing state to 'StandStill':

- **MC_HomeDirect** - Static Homing forcing a position from a user reference;
- **MC_HomeAbsolute** - Static Homing forcing a position from an absolute encoder;
- **MC_FinishHoming** - Finishes the homing procedure with the possibility to do a relative move to the operational area.

In addition, the homing functionality is needed while the machine is operating, i.e. not in the state 'Homing'. This 'homing-on-the-fly' is called passive homing. They have no effect on State Diagram, like the administrative FBs, and can be called in any movement state. They consist of:

- **MC_StepReferenceFlyingSwitch**
- **MC_StepReferenceFlyingRefPulse**
- **MC_AbortPassiveHoming**

XI. 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:

XII. Conclusion

The publication of the specification and the first implementations clearly showed that the multi-axes implementation certainly fits within the framework as defined by the IEC 61131-3 standard.

With many implementations becoming available, 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 targets which can be selected in a later 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, the specification itself, as well as information on the contributions of PLCopen in safety, communication and exchange, check www.PLCopen.org.