Release Notes for Maestro

Version X.1.7.0

This guide is delivered subject to the following conditions and restrictions:

- This guide contains proprietary information belonging to Elmo Motion Control Ltd. Such information is supplied solely for the purpose of assisting users of the Gold Line technology.

- The text and graphics included in this manual are for the purpose of illustration and reference only. The specifications on which they are based are subject to change without notice.

- Information in this document is subject to change without notice.

Revision History

Version	Date	Details
Ver. 1.000	May. 2017	Initial version

Release Notes - New Firmware Version for Maestro Version x.1.7.0 and Library Update 275

1. General

The "***uImage_vX.1.7.0_B01_2017_04_05.Xms***" is a new firmware release for the Gold and Platinum Master Motion Controller.

The major x.x.x.x version. When it is 1 (***uImage_v1.1.7.9_B01_2017_04_05.gms)*** – it is a GMAS version. When is is 2 – it is Platinum(***uImage_v2.1.7.0_B01_2017_04_05_IEC_1.0.0.3.pms)***.

The GMAS and Platinum versions will always be released together.

New libraries were released as well. Version 275 of the libraries include an additional directory that is specific to the Platinum Maestro. This is under GMAS\lib\platinum. As far as API's, include directories – it is 100% compatible with the existing Gold Maestro.

The main issues of this release are the following:

- New IEC programming Environment
- SIL – *Software In The Loop*
- Profiler Simulator now released in win32 and .NET interfaces.
- Full support for download FoE at 250us cycle time.

Known Issues:

- Inserting a Stop command when there are more than 150 Function Blocks in the Motion Queue (whether the axis is a Single or Group axis) can cause a crash in the **Platinum Maestro**. There are no known issues when there are less than 150 Function Blocks in the motion queue.

1.1 Enhanced IEC programming Environment – PMAS Only

This version now supports an additional and enhanced IEC programming environment. In addition to the legacy environment supported in the EAS – we now support the Codesys programming environment.

The legacy environment will be supported for approximately 1 year from this release.

The EAS identifies which environment the user is working in, and suggests the new environment. This is done by reading the IEC_RT_TYPE_ACT parameter from the Maestro. If it is 1 (enhanced) , starting the runtime shall start the Codesys runtime. Otherwise the legacy runtime will be started.

The user will be recommended to use the new environment:

Startng / stopping the runtime will be via the same interface:
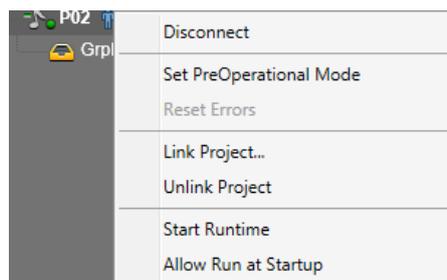
Please refer to the following document in order to program in the new IEC environment:

XXXXX

1.2 Enhanced IEC Version and Packages

The new Firmware includes the legacy IEC and the enhanced IEC within the firmware image.

The Enhanced IEC version is stated in the firmware version. This is the identifier of the IEC version.

<p align="center">***uImage_v2.1.7.0_B01_2017_04_05_IEC_1.0.0.3.pms***</p>

In the case above – the version is 1002. This version identifies the 'Package' that is to be used on the IDE side.

When installing a package (always released together with a firmware version) – the following is installed:

- Elmo Motion Control Devices
- Elmo Motion Control Components.
- Documentation.

The user does not need to be confused as to what is supported and where. The IEC runtime matches the package release. In the future – when updated packages are released – the previous package can be uninstalled via the *Package Manager.*

Elmo Motion Control Devices

The devices supported are the following:

Elmo Platinum Maestro Device (The Hard Realtime PLC).

1. Elmo Master Device (Scanner Device).
2. Elmo Modbus
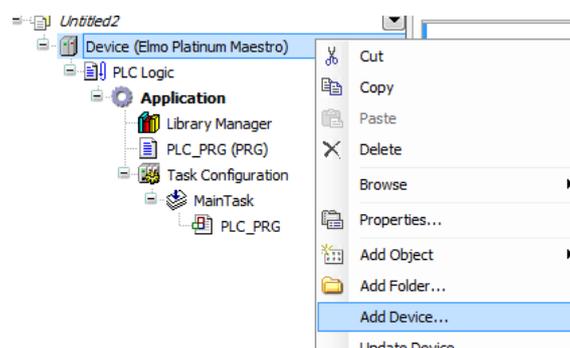
After creating a new project – by default – the project includes the Elmo Platinum Maestro together with an *Elmo Master Device*

Scanning For Devices shall return the Ethercat configuration with the names and Axis References of the Ethercat slaves. Via this configuration – one can map variables to IO's.
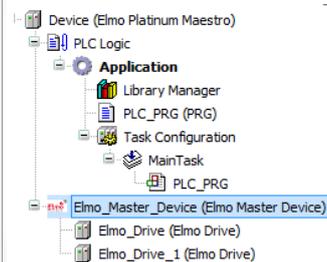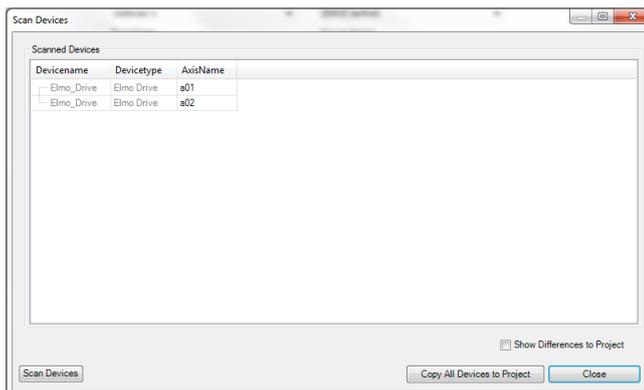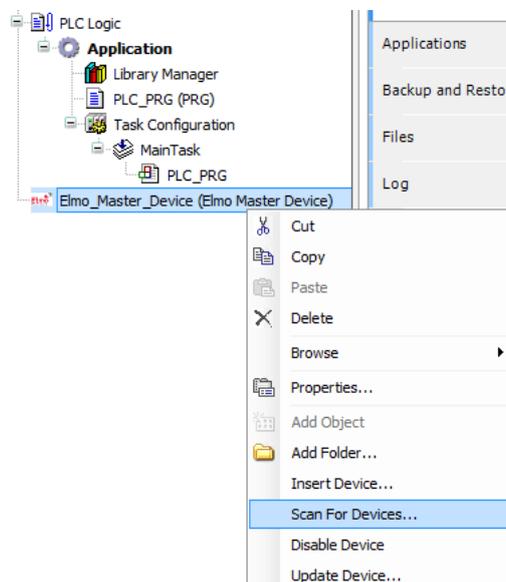
Elmo Modbus Device

Another device type is the Modbus device. By adding this component – the user can set Read/Write memory regions within the Maestro and map specific Modbus variables to program parameters
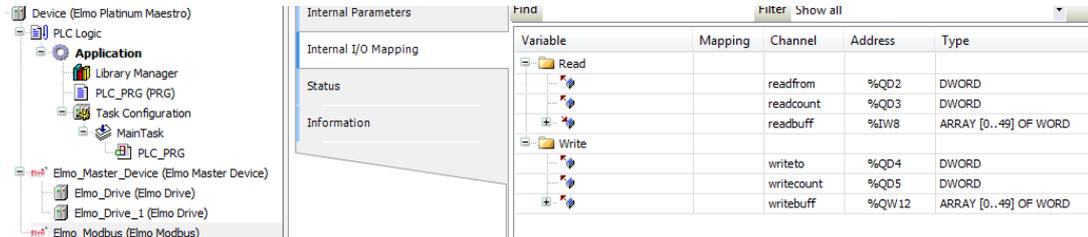
Elmo Motion Control Components

Elmo's components are automatically installed together with the package. In order to add the component – the user must add the relevant library to the project – from the Codesys repository (data base).

The following libraries / components are available:
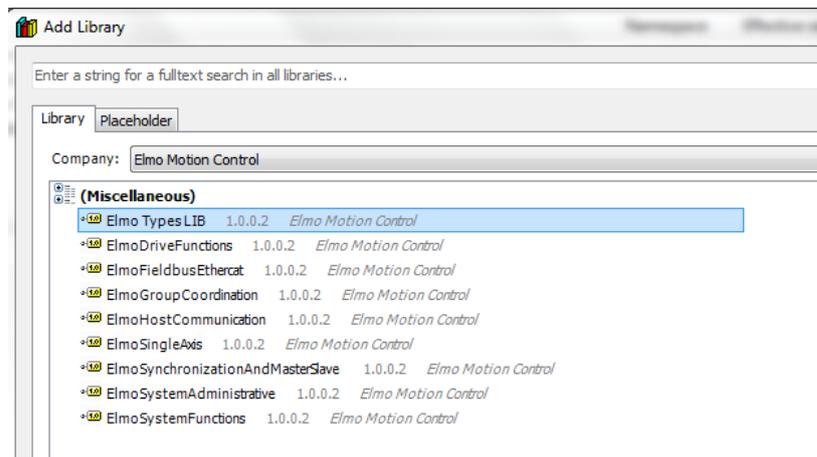
1. Single Axis – Functions, Functin Blocks, EMBLS related to single axis.
   - Motion
   - Homing
   - Axis Functions
   - Position to Force
   - Axis Status and Parameters
2. Group Coordination
   - Homing
   - Axis Functions
   - Position to Force
   - Axis Status and Parameters
   - Motion
   - Homing
   - Raster Scanning Motions
   - Transformations
   - Group Statuses and Parameters
   - Group Axis Functions
   - Synchronized
3. Synchronization and Master Slave
   - Superimposed
   - ECAM
   - Gearing
   - JoyStick
   - Trajectory
4. Fieldbus Ethercat
   - Configurations
   - Process Image
   - Diagnostics
5. ELMO Drive Functions
   - FeedBack Emulation
   - Output Compare
   - Binary Interperter
   - EOE
   - COE
   - Digital IOs
6. Host Communication
   - Modbus
7. System Functions
   - Error correction
   - System OC
   - TouchProbe
   - Motion Queue
   - State Conditional Waits
8. System Administrative

- Statuses and Parameters
- System Functions
- Error Policies
- Error handling
- BulkRead

Please see Appendix of this version for the **FULL** Function Block and Function list.

Documentation

From within the library manager – the user can select the relevant Function Block and see the relevant documentation:
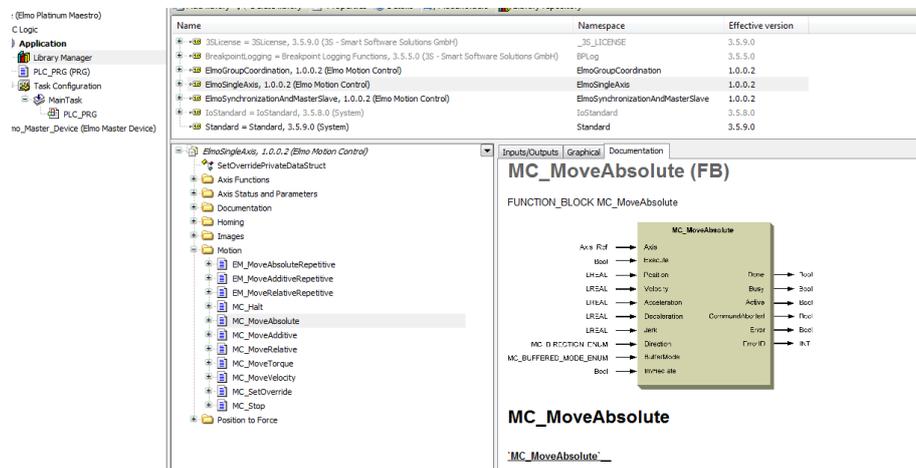
Function Naming Conventions

- If the function is part of the PLCOpen standard – the function name shall remain as-is.
- Elmo Functions i.e. HomeDS402 - which is an Elmo FB – shall have a prefix of EM_
- For EMBL's (Elmo Motion Block Library) Functions – Shall have an EML_ prefix.

1.3 Enhanced IEC – Highlight Features

As part of the new IEC environment – there are several highlights that are 'Built In' IDE features and features that were developed by Elmo:

- Events from Maestro – Events are easily configured. Project templates shall include examples for supporting events.

- Multi Tasking – A built in feature of the Enhanced IEC environment.
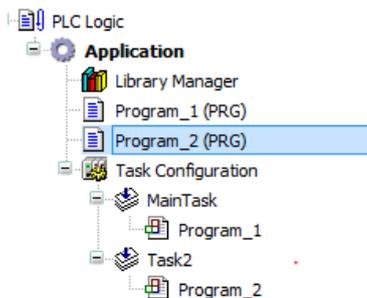
  We can have 2 programs. Each program can run from a different task. Each task can be assigned a different priority and type:

- Deterministic Task – A deterministic task – AKA *External* - is one of the 'Task Types'. This task runs at the RealTime core, and is executed at the end of the cycle call.
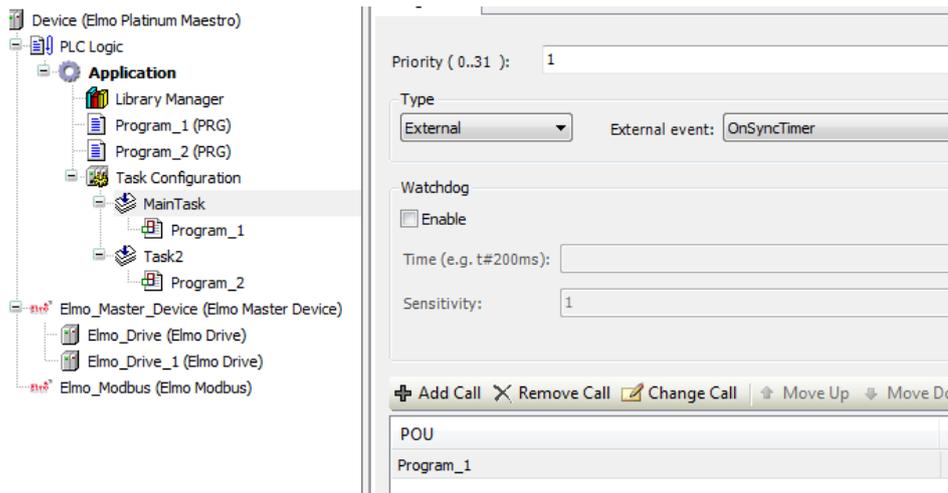
- Fieldbus Topology Scanner – The capability of scanning <u>a given network</u> (Does not replace the EAS Ethercat Configuration Tool). In addition – The topology tree shows whether I am in Sync with the configuration or not.
- Modbus Device – A great method for mapping network variables to local program variables.
- Web Visualization – One of the main reasons for the move to the Enhanced IEC environment is the support for Web (From a browser, phone or tablet) visualization and Target visualization (to be supported in the future).
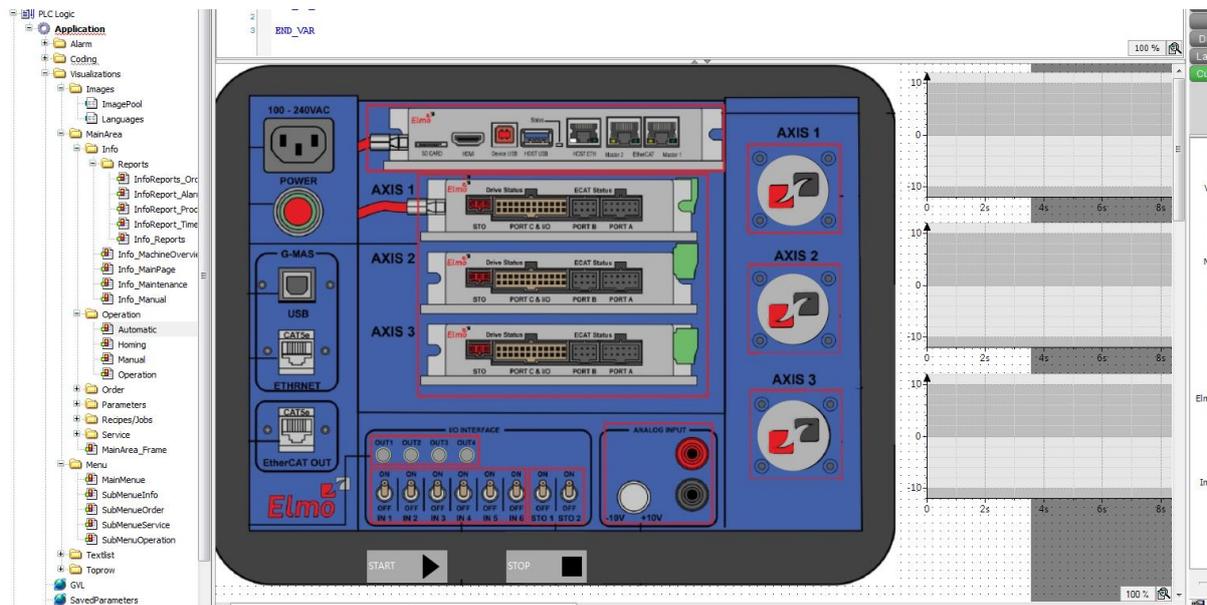
Creating an 'EAS like' screen, with 3 x Single Axes – is easy after a template for ONE Single axis was created.
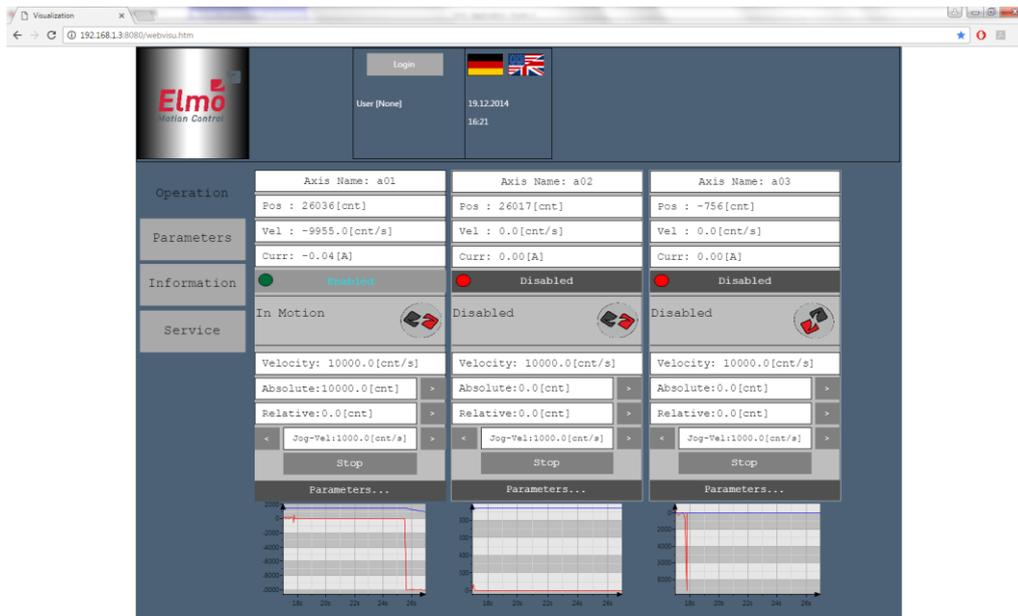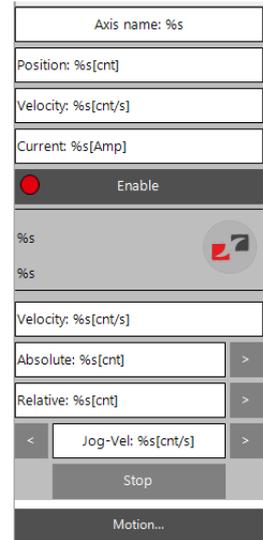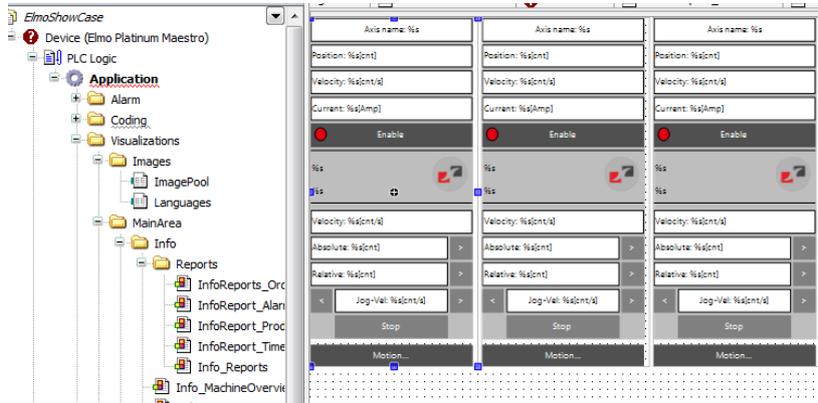
-   Multi Language Support – A built in option in Codesys is the method they handle Lists of Strings according to languages. Changing languages is performed very easily.

1.4   SIL – *Software In the Loop*

Software-in-the-loop (SIL), is a technique that is used in the development of complex real-time embedded systems. SIL provides an effective platform by adding the complexity of the plant / algorithm under control to the system platform. The implementer can easily connect and run these plants / algorithms within the Platinum Maestro.

Whether the implementer wishes to create his own control loop, or run his own profiler or even implement his own Kinematic equations for Robot Kinematics – this can now be accomplished easily with Elmo's Platinum Maestro.

In order for SIL to run, the user needs to run the code in a deterministic manner.

There are two main cores in the system:

- Operating System Core – This core handles all Operating System tasks. Communication stacks. File System. Etc…
- Real Time Core – The only code that runs on this core is affiliated to the Maestro Real Time. Profilers. Fieldbus. Etc…
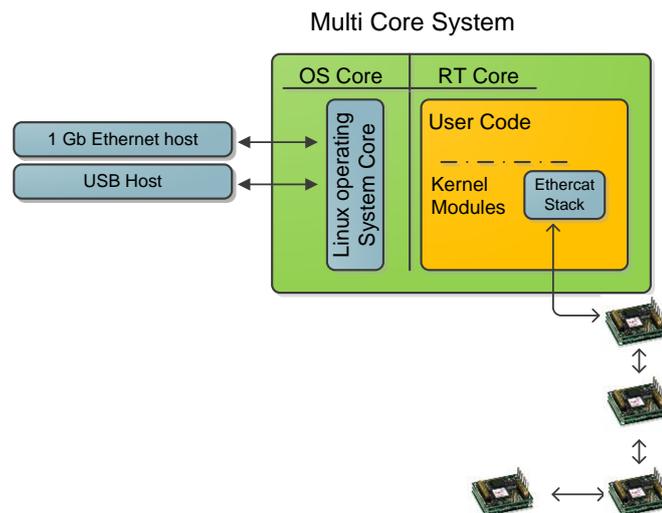
The user now has the capability of using the generated code from the Mathworks Embedded Coder®, combining it to a Platinum Maestro 'User Program', and running the generated code in *Real Time Context*, thus promising that the code will run deterministically.

The generated code can be either from Matlab or Simulink Coders – or code that was written by the programmer.

The functionality that can run in SIL is limited. The user cannot run regular API's under SIL – as they are very time consuming. The idea behind SIL is – the user has an algorithm with direct Inputs and Outputs to the fieldbus – SIL allows the user, with a specific set of API's, to Read/Write to the fieldbus.
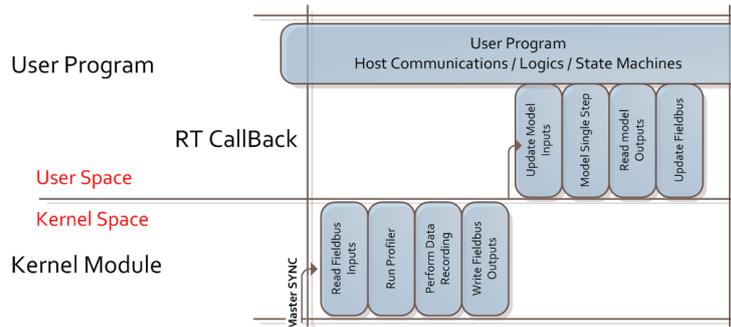
Our Solution

Elmo's Platinum Maestro communicates with the Elmo Drives via the CiA DS-402 Drive Profile protocol.

This protocol defines the Information that is to be exchanged between the Master Network Controller (Maestro) and the Slaves (Servo Drives).

For instance, if we take the *Cyclic Synchronous Position (CSP)* Mode:
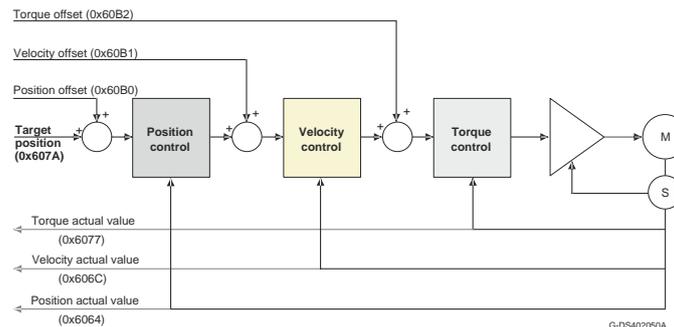
Every control loop (Currently down to 250us in the Platinum Maestro when running 16 drives), when the Maestro works in *Cyclic Synchronous Position (CSP)* Mode, the following information is exchanged:

Input Objects:

- Torque Actual Value
- Velocity Actual Value
- Position Actual Value

Output Objects:

- Torque Offset.
- Velocity Offset.
- Position Offset.
- Target Position.

In addition, the user may run in *Cyclic Synchronous Velocity* or *Cyclic Synchronous Torque* modes.

Together with allowing FULL access to the Platinum Maestro Real Time internal 64bit double precision FP axis variables and Fully Deterministic *Real Time* callback functionality and the capability of directly writing to the Fieldbus (AKA the *Process Data)* – the user can easily create his own profiler, Kinematics or algorithm – on top of – or alongside to the Platinum Maestro's Profiler.

Specific API's were added to the Platinum Maestro Library, enabling fast access to the *Process Data* Variables.

The *Real Time* Periodic code of the model is to be called in the predefined *RealTime* callback. The user is responsible for linking the model variables to the *Platinum Maestro Process Image* variables. Due to the vast support for 64 bit Floating Point precision and the full support for mathematical libraries in the *Platinum Maestro* - basically any model can be executed. The user is of course notified, and can read statuses regarding the execution time of the created model, and is granted external access to the model's parameters even during the execution of the model.

The advanced data recording capabilities of the *Platinum Maestro* are not left astray. The user may perform data recording on the internal model variables, in addition to the *Process Image* parameters enabling the user to closely debug and synchronize system signals, together with the model signals.

Basically – what we do is – we give the user the capability and the responsibility of updating the fieldbus. Instead of the profiler Real Time being the one responsible for updating the fieldbus – the user is responsible for this.

New APIs and libraries

Name	Type	Data Rec	Parameter access	Save?	Special Function	Limits	Default
lUser607a	Long32	Yes	R/W	No	No	None	0
lUser60b0	Long32	Yes	R/W	No	No	None	0
lUser60ff	Long32	Yes	R/W	No	No	None	0
lUser60b1	Long32	Yes	R/W	No	No	None	0
sUser6071	Short16	Yes	R/W	No	No	None	0
sUser60b2	Short16	Yes	R/W	No	No	None	0
ucUser607a_Src	Uchar8	No	R/W	Yes	Yes	0-2	eSrc_NC
ucUser60b0_Src	Uchar8	No	R/W	Yes	Yes	0-2	eSrc_ZERO
ucUser60ff_Src	Uchar8	No	R/W	Yes	Yes	0-2	eSrc_NC
ucUser60b1_Src	Uchar8	No	R/W	Yes	Yes	0-2	eSrc_NC
ucUser6071_Src	Uchar8	No	R/W	Yes	Yes	0-2	eSrc_NC
ucUser60b2_Src	Uchar8	No	R/W	Yes	Yes	0-2	eSrc_ZERO
lUserMB1	Long	Yes	R/W	No	No	None	0
lUserMB2	Long	Yes	R/W	No	No	None	0

All XX_*Src* suffixed parameters, state the source of the PI output DS402 parameters. The ePIOutSrc has the following values:

0 - eSrc_NC ➔ the PI source is the NC profiler outputs. For instance – the 0x607a source is the Profiler Output parameter.

1 - eSrc_ZERO ➔ the source points to an address that includes a ZERO (0) value.

2 - eSrc_USER ➔ the source is the corresponding user parameter source.

Access to these parameters are via the Get/Set parametesrs interface.

When the source is eSrc_USER – it is under the user's responsibility to update the fieldbus. This is done via the new CMMCRTSingleAxis class.

This class derives from the CMMCSingleAxis class. In addition has the following methods:

Setting Fieldbus DS402 Objects:

```
SetUser607A(int iUser607Aval)
SetUser60FF(int iUser60FFval)
SetUser60B1(int iUser60B1val)
SetUser60B2(double iUser60B2val)
SetUser6071(double dUser6071val)
SetUserMB1(int iUserMB1)
SetUserMB2(int iUserMB2)
SetUserCW(unsigned short usUserCWval)
SetDigitalOutPuts(unsigned int uiDigitalOutputs)
```

Retrieving Fieldbus DS402 Objects and internal profiler parameters:

```
int GetUser607A()
int GetUser60FF()
int GetUser60B1()
double GetUser60B2()
double GetUser6071()
int GetUserMB1()
int GetUserMB2()
int GetActualPosition()
double GetActualVelocity()
double GetdPos()
double GetdVel()
short GetAnalogInput()
unsigned int GetDigitalInputs()
double GetActualCurrent()
unsigned int GetPLCOpenStatus()
unsigned short GetControlWord()
unsigned short GetStatusWord()
unsigned long GetCycleCounter()
```

1.5 Profiler Simulator

A new feature has been introduced to EAS 2.3.0.7. This feature is the capability of running and simulating motion profiles on the EAS, without any hardware.

The following is supported:

- Single Axis motion
- Group Axis Motion
- Table affiliated motion
  - Splines
  - PVT.
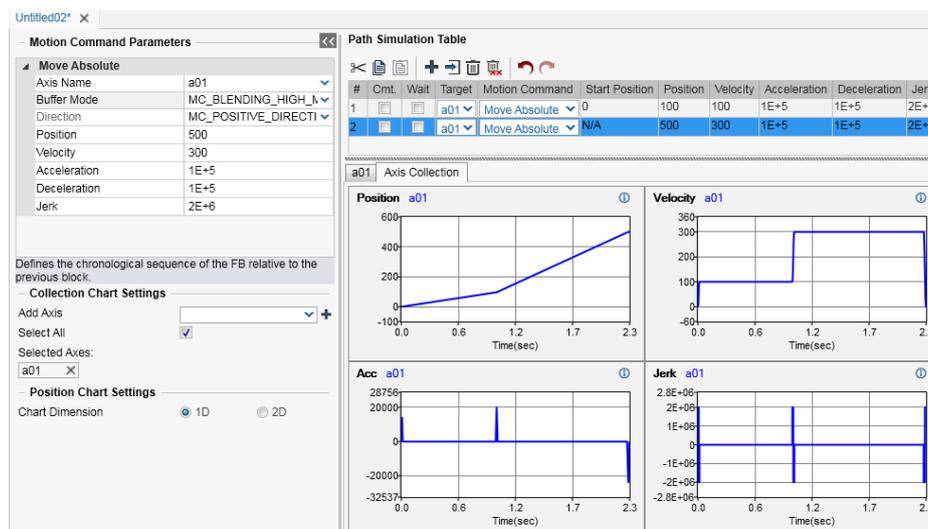
This feature uses a .NET interface which incorporates the identical profiler code that runs on the Maestro.

The motion API used is the identical API that is used when working with the Maestro. the difference is – the connection type.

A new connection has been defined:

Create connection :(Get instance of the class `MMCProfilerSimulator`) :

```
MMCProfilerSimulator ConnectProfilerSimulator(out int hndl, bool throwExceptionOnNotSupportedFunction
=false)
```

The user needs to initialize each motion via a dedicated Init function, and then call the relevant motion commands.

Single Axis example:

```
int res = 0;
int hndl;
MMCProfilerSimulator profiler = MMCConnection.ConnectProfilerSimulator(out hndl);

MMCSingleAxis a01 = new MMCSingleAxis("a01", hndl);
a01.Acceleration = 1000000;
a01.Deceleration = 1000000;
a01.Velocity = 50000;
```

```
a01.Jerk = 10000000;
a01.Execute = 1;


MMCProfilerSimulatorConfiguration singleConfiguration = new MMCProfilerSimulatorConfiguration();
singleConfiguration.NodeType = NC_NODE_TYPE_ENUM.NC_NODE_SING_AXIS_TYPE; // single axis configuration
singleConfiguration.NumOfFB = 1; // there is 1 FB in queue

singleConfiguration.StartPos[0] = 0;
singleConfiguration.RV[0] = PROF_SIM_RECORDER_VALS_ENUM.PROF_SIM_RECORDER_DESIRED_POS;
singleConfiguration.RV[1] = PROF_SIM_RECORDER_VALS_ENUM.PROF_SIM_RECORDER_DESIRED_VEL;

res = profiler.Init(singleConfiguration);
if (res < 0)
        Console.WriteLine("Failed to init Simulator. error= " + (MMCErrors)res);

res = a01.MoveAbsolute(5000, MC_BUFFERED_MODE_ENUM.MC_BUFFERED_MODE);
bool isProfilerFinished;
uint ActualCycleTime;
res = profiler.Run(-1, out isProfilerFinished, out ActualCycleTime);
if (res < 0)
        Console.WriteLine("Failed to Run profiler. error = " + (MMCErrors)res);


double[][] recordingData;
double[] endPos;
res = profiler.GetDataRecording(out recordingData, out endPos);
if (res < 0)
        Console.WriteLine("Failed to Get data recording. error = " + (MMCErrors)res);

return;
```

Group Axis example:

```
int res = 0;
int hndl;
MMCProfilerSimulator profiler = MMCConnection.ConnectProfilerSimulator(out hndl);

MMCGroupAxis v01 = new MMCGroupAxis("v01", hndl);
v01.Acceleration = 1000;
v01.Deceleration = 10000;
v01.Velocity = 50000;
v01.Jerk = 10000;
v01.CoordSystem = MC_COORD_SYSTEM_ENUM.MC_MCS_COORD;
v01.TransitionParameter = new double[16];
v01.Execute = 1;
v01.TransitionMode = NC_TRANSITION_MODE_ENUM.MC_TM_NONE_MODE;
v01.Superimposed = 0;
v01.EndPoint[0] = 10000;
v01.EndPoint[1] = 20000;
v01.EndPoint[2] = 0;
v01.AuxPoint[0] = 100;
v01.AuxPoint[1] = 200;
v01.AuxPoint[2] = 5000;
v01.TransitionParameter[0] = 0;
v01.TransitionParameter[1] = 0;
v01.TransitionParameter[2] = 0;
v01.TransitionMode = NC_TRANSITION_MODE_ENUM.MC_TM_NONE_MODE;

MMCProfilerSimulatorConfiguration groupConfiguration = new MMCProfilerSimulatorConfiguration();
groupConfiguration.CoordSystem = MC_COORD_SYSTEM_ENUM.MC_MCS_COORD;
groupConfiguration.NodeType = NC_NODE_TYPE_ENUM.NC_NODE_MULT_AXIS_TYPE;
groupConfiguration.NumAxesInGroup = 2;
groupConfiguration.NumOfFB = 1;
groupConfiguration.MaxAC = 10000;
groupConfiguration.MaxVel = 10000;
groupConfiguration.MaxDC = 10000;
groupConfiguration.StartPos[0] = 0;
groupConfiguration.StartPos[1] = 0;
groupConfiguration.CoordAxes = NC_MC_COORD_AXES.NC_XY_AXES;
groupConfiguration.RV[0] = PROF_SIM_RECORDER_VALS_ENUM.PROF_SIM_RECORDER_DESIRED_POS_X;
groupConfiguration.RV[1] = PROF_SIM_RECORDER_VALS_ENUM.PROF_SIM_RECORDER_DESIRED_POS_Y;

res = profiler.Init(groupConfiguration);
if (res < 0)    Console.WriteLine("Failed to init Simulator. error= " + (MMCErrors)res);

uint ures = v01.MoveLinearAbsolute(10000, new double[] { 5000, 2000, 3000, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
MC_BUFFERED_MODE_ENUM.MC_BUFFERED_MODE);

 bool isProfilerFinished;
```

```
uint ActualCycleTime;
res = profiler.Run(-1, out isProfilerFinished, out ActualCycleTime);
if (res < 0)    Console.WriteLine("Failed to Run profiler. error = " + (MMCErrors)res);


double[][] recordingData;
double[] endPos;
res = profiler.GetDataRecording(out recordingData, out endPos);
if (res < 0) Console.WriteLine("Failed to Get data recording. error = " + (MMCErrors)res);

return;
```

Appendix

1. Single Axis – Functions, Functin Blocks, EMBLS related to single axis.
   - Motion
     - MC_Halt
     - MC_MoveAbsolute
     - MC_MoveAdditive
     - MC_MoveRelative
     - MC_MoveTorque
     - MC_MoveVelocity
     - MC_Stop
     - MC_SetOverride
   - Homing
     - EM_HomeDS402
     - EM_HomeDS402Ex
     - EML_HomeImmediate
     - EML_HomeOnBlock
     - EML_HomeOnHomeAndLimitSwitch
     - EML_HomeOnIndex
     - EML_HomeOnLimitSwitch
   - Axis Functions
     - MC_Power
     - EM_ChangeOpMode
     - MC_Reset
     - MC_ChangeOpModeEX
   - Position to Force
     - EML_PositionForce
     - EML_PositionForceXYZ
   - Axis Status and Parameters
     - EM_GetAxisRef
     - MC_ReadActualPosition
     - MC_ReadActualTorque
     - MC_ReadActualVelocity
     - MC_ReadAxisError
     - EM_GetOpMode
     - MC_ReadStatus
     - EML_StatusRegisterIsFLS

- o EML_StatusRegisterIsRLS
- o EML_StatusRegisterIsSWLimitHigh
- o EML_StatusRegisterIsSWLimitLow
- o EML_StatusRegisterIsTargetReached
- o EML_SPLCOpenIsDisabled
- o EML_SPLCOpenIsError
- o EML_SPLCOpenIsInMotion
- o EML_SPLCOpenIsStandStill
- o MC_ReadBoolParameter
- o MC_ReadParameter
- o MC_WriteBoolParameter
- o MC_WriteParameter
- o EML_SetModulo
- o EML_SetUserUnits

2. Group Coordination
   - Motion
     - o MC_MoveCircularAbsolute
     - o EM_MoveCircularAbsoluteAngle
     - o EM_MoveCircularAbsoluteBorder
     - o EM_MoveCircularAbsoluteCenter
     - o EM_MoveCircularAbsoluteRadius
     - o obsolete
     - o MC_MoveLinearAbsolute
     - o MC_MoveLinearAdditive
     - o MC_MoveLinearRelative
     - o obsolete
     - o EM_MovePolynomAbsolute
     - o MC_GroupStop
     - o EML_SimpleXY
     - o EML_SimpleXYZ
     - o MC_GroupSetOverride
     - o MC_GroupHalt
   - Homing
     - o EML_MHomeImmediate
     - o EML_MHomeOnBlock
     - o EML_MHomeOnHomeAndLimitSwitch
     - o EML_MHomeOnIndex
     - o EML_MHomeOnLimitSwitch
   - Raster Scanning Motions
     - o EML_ScanRasterXYZ
     - o EML_ScanRasterXY
   - Transformations
     - o EM_SetDeltaRobotKinematics

- o EM_SetKinTransform
- o MC_SetScaraRobotKinematics
- o EML_Simple2DKinematics
- o EML_Simple3DKinematics
- o MC_SetCartesianTransform
- Group Statuses and Parameters
  - o EM_GetGroupAxisRef
  - o MC_GroupReadStatus
  - o MC_GroupReadActualPosition
  - o EM_GetGroupMembersInfo
  - o EM_GroupReadBoolParameter
  - o EM_GroupReadParameter
  - o EM_GroupWriteBoolParameter
  - o EM_GroupWriteParameter
  - o EML_MPLCOpenIsDisabled
  - o EML_MPLCOpenIsError
  - o EML_MPLCOpenIsInMotion
  - o EML_MPLCOpenIsStandby
  - o MC_GetStatusRegister
- Group Axis Functions
  - o MC_GroupDisable
  - o MC_GroupEnable
  - o MC_GroupReset
  - o MC_AddAxisToGroup
  - o MC_RemoveAxisFromGroup
- Synchronized
  - o MC_TrackRotaryTable
3. Synchronization and Master Slave
   - Superimposed
     - o EM_AxisLink
     - o EM_AxisUnLink
   - ECAM
     - o MC_CAMIn
     - o MC_CAMOut
     - o EM_CAMStatus
     - o EM_CAMTableAdd
     - o EM_CAMTableInit
     - o MC_CAMTableSelect
     - o EM_CAMTableSet
     - o EML_CAMMemoryActivate
     - o EML_CAMMemoryInit
     - o EML_CAMXMLActivate
   - Gearing
     - o MC_Gear_In

- JoyStick
  - o EML_JoystickPosition
  - o EML_JoystickPositionAdvanced
  - o EML_JoystickPositionExternalParam
  - o EML_JoystickPositionPI
  - o EML_JoystickVelocity
  - o EML_JoystickVelocityAdvanced
  - o EML_JoystickVelocityExternalParam
  - o EML_JoystickVelocityPI
- Trajectory
  - o EM_GetTableIndex
  - o EM_InitTable
  - o EM_LoadTableFromFile
  - o EM_UnloadTable
  - o EM_AppendPointsToTable
  - o MC_PathSelect
  - o MC_PathUnselect
  - o MC_MovePath
  - o EM_MoveTable
  - o EML_PVTFILE
  - o EML_PVTMEM
4. Fieldbus Ethercat
   - Configurations
     - o EM_ConfigPIBulkRead
     - o EM_PerformPIBulkRead
   - Process Image
     - o EM_GetPIInfo
     - o EM_GetPIInfoByAlias
     - o EM_ReadPIByIndex
     - o EM_ReadPIRAWByIndex
     - o EM_WritePIByIndex
     - o EM_WritePIRAWByIndex
   - Diagnostics
     - o EM_GetCommDiagnostics
     - o EM_GetCommStatistics
     - o EM_GetEthercatCommStatistics
     - o EM_ResetCommDiagnostics
     - o EM_ResetCommStatistics
5. ELMO Drive Functions
   - FeedBack Emulation
     - o EML_FeedbackPWMSignalEmulation
     - o EML_FeedbackPWMSourceSocketEmulation
     - o EML_FeedbackQuadEmulation
     - o EML_FeedbackQuadSignalEmulation

- o EML_FeedbackWaveSignalEmulation
- Output Compare
  - o EML_OCAbsolutePositionActivate
  - o EML_OCAbsolutePositionInit
  - o EML_OCPositionTabulatedActivate
  - o EML_OCPositionTabulatedInit
  - o EML_OCSinglePulseActivate
  - o EML_OCSinglePulseInit
  - o EML_OCTimeTabulatedActivate
  - o EML_OCTimeTabulatedInit
- Binary Interperter
  - o EM_ElmoGetFloatArray
  - o EM_ElmoGetFloatParam
  - o EM_ElmoGetIntArray
  - o EM_ElmoGetIntParam
  - o EM_ElmoSetFloatArray
  - o EM_ElmoSetFloatParameter
  - o EM_ElmoSetIntArray
  - o EM_ElmoSetIntParameter
  - o EM_ElmoCall
  - o EM_ElmoExecute
- EOE
  - o EM_EoEClose
  - o EM_EoEConnect
  - o EM_EoEGetParam
  - o EM_EoESetArray
  - o EM_EoESetParam
  - o EM_EoEGetArray
- COE
  - o EML_ReadElmoArrayCoE
  - o EML_ReadElmoCmdCoE
  - o EML_ReadElmoParamCoE
  - o EML_WriteElmoArrayCoE
  - o EML_WriteElmoCmdCoE
  - o EML_WriteElmoParamCoE
- Digital IOs
  - o MC_ReadDigitalInput
  - o EM_Read32DigitalInputs
  - o EM_Write32DigitalOutputs
6. Host Communication
  - Modbus
    - o EM_ModbusReadCoilsTable
    - o EM_ModbusWriteHoldingRegisterTable
    - o EM_ModbusReadInputsTable

- o EM_ModbusStopServer
- o EM_ModbusStartServer
- o EM_ModbusWriteHoldingRegisters
- o EM_ModbusIsRunning
- o EM_ModbusWriteCoilsTable
- o EM_ModbusReadHoldingRegisterTable
- o EM_ModbusReadHoldingRegisters
7. System Functions
  - Error correction
    - o EM_DisableErrorCorrTable
    - o EM_EnableErrorCorrTable
    - o EM_LoadErrorCorrTable
    - o EM_UnloadErrorCorrTable
    - o EM_GetErrorTableStatus
  - TouchProbe
    - o EMC_TouchProbeDisable
    - o EMC_TouchProbeEnable
  - Motion Queue
    - o EM_WaitUntilConditionEx
    - o EM_WaitUntilCondition
    - o EM_Dwell
    - o EM_GetFbDepth
    - o EM_GetTotalFbDepth
  - State Conditional Waits
    - o EML_WaitInMotionMask
    - o EML_WaitMotionSettled
    - o EML_WaitMotorOn
    - o EML_WaitOperationalMode
    - o EML_WaitSpecificMask
    - o EML_WaitStandStill
8. System Administrative
  - Statuses and Parameters
    - o EM_GlobalReadBoolParameter
    - o EM_GlobalReadParameter
    - o EM_GlobalWriteBoolParameter
    - o EM_GlobalWriteParameter
    - o EM_WriteGroupOfParameters
  - System Functions
    - o EM_ResetSystem
  - Error Policies
    - o EM_GetErrPolicy
    - o EM_RegErrPolicy
  - Error handling
    - o EM_GetErrorDescription

- o EM_GetLastError
- BulkRead
  - o EML_ActivateBulkReadNonpreset
  - o EML_ActivateBulkReadPreset1
  - o EML_ActivateBulkReadPreset2
  - o EML_ActivateBulkReadPreset3
  - o EML_ActivateBulkReadPreset4
  - o EML_ActivateBulkReadPreset5
  - o EML_InitBulkReadByPerset
  - o EML_InitBulkReadNonPerset

Release Notes - New Firmware Version for Maestro Version x.1.6.1 and Library Update 271

2. General

The "***uImage_vX.1.6.1_B06_2016_03_31.gms***" is a new firmware release for the Gold and Platinum Master Motion Controller.

This is the first official Platinum Maestro Release. As far as features are concerned, the Platinum Maestro supports all of the features of the Gold Maestro.

In addition to the standard features of the Gold Maestro we added the capability of setting the Platinum Maestro time (there is a RT Clock and battery backup in the Platinum Maestro).

The major x.x.x.x version. When it is 1 (***uImage_v1.1.6.1_B06_2016_03_31.gms)*** – it is a GMAS version. When is is 2 – it is Platinum(***uImage_v2.1.6.1_B06_2016_03_31.pms)***.

The GMAS and Platinum versions will always be released together.

As the Platinum Maestro is based on a much faster CPU platform (Dual Core 1.5 GHz), the number of axes supported (and tested), and the Cycle Time we can reach differ from the Gold Maestro.

The following lists the supported Cycle Times and number of axes supported in the Platinum Maestro:

Number of Axes	Cycle Time (uSec)
<=16	250us
<=32	500us
<=64	1000us
<=96	2000us

New libraries were released as well. Version 271 of the libraries include an additional directory that is specific to the Platinum Maestro. This is under GMAS\lib\platinum. As far as API's, include directories – it is 100% compatible with the existing Gold Maestro.

Since the Gold maestro and the Platinum Maestro are both based on different platforms and Endianess – the .NET API and the C/CPP libraries from the host (RPC) read the platform that is targeted and performs endianness swapping according to the target platform.

When compiling code that is to run on the target (via the new MDS – **M**aestro **D**eveloper **S**tudio) – a new toolchain for the Platinum Maestro is available. Please see the MDS documentation for further information.

Known Issues:

- CAN in Interpolated mode Issues.

Ethercat Hotplug issues.

Release Notes - New Firmware Version for Maestro Version 1.1.6.0 and Library Update 267

3. General

The "*uImage_v1.1.6.0.B2_.gms*" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. Online Splines
2. New ECAM Capabilities
   - 2.1 Modifying ECAM properties, on the fly.
   - 2.2 New ECAM Interpolation types.
3. UVW Rotational Axes Support
4. New Kinematics Supported
   - 4.1 Scara Robot Kinematics
   - 4.2 3 Link Robot Kinematics
5. PCS Transformation Support
6. Track Rotary Table Support
7. Data Logging
8. Beckhoff ADS Protocol Support
9. Preparations for Platinum Release
10. EMBL's
11. New API's
12. Bug Fixes

3.1 Online Splines

In addition to the existing Offline Splines and PVT we now introduce 'Online Splines'.

Online Splines – or – PT (Position Time), is similar to PVT:

- Can be loaded from a file or via memory.
- Supported for Single and Group axes.
- On – the – Fly points update.
- Ability to define Underflow threshold event.

There are numerous Online Spline modes:

- MC_QUINTIC_ON_PARAB_FT_DWELL: *Fixed time*.
- MC_QUINTIC_ON_PARAB_VT_DWELL: *Variable time*.
- MC_QUINTIC_ON_PARAB_CV_DWELL: *Constant velocity*.

The Online Splines feature has similar API's to PVT:

MMC_InitTableCmd	Allocates a memory segment in the shared memory according to the dimension and number of points(same as PVT). Using default values for constant velocity and fixed time.
MMC_InitTableExCmd	Same as MMC_InitTableCmd with additional input parameters for constant velocity and fixed time.
MMC_AppendPointsToTableCmd	Appends (or modifies) points to(in) an existing table (same as PVT).
MMC_UnloadTableCmd	Unload table from journal on Maestro (same as PVT).
MMC_MoveTable	Move axis system (dynamic or not) according to loaded table (same as PVT).

The compatible functions in IEc and CPP remain intact.

An example for an 'Online spline' table:

```
PT mode          8
PT dimension     2                      Group Axis of 2 axes
PT num of pts    10                     10 points in table
PT cyclic        1                      Table is cyclic
PT pos absolute  1                      Positions are absolute
PT time absolute        0               Time are NOT absolute
PT spline type   3                      Spline Type = CV
PT const velocity       10000           Constant Velocity  = 10000
PT fixed time    3
PT data start
Time    P1     P2
1       10000  20000
1       20000  30000
1       30000  40000
1       40000  50000
1       50000  60000
1       60000  50000
1       50000  40000
2       40000  30000
2       30000  20000
1       20000  10000
PT data end
```

The labels in RED are not to be part of the file

Running this will produce the following chart:
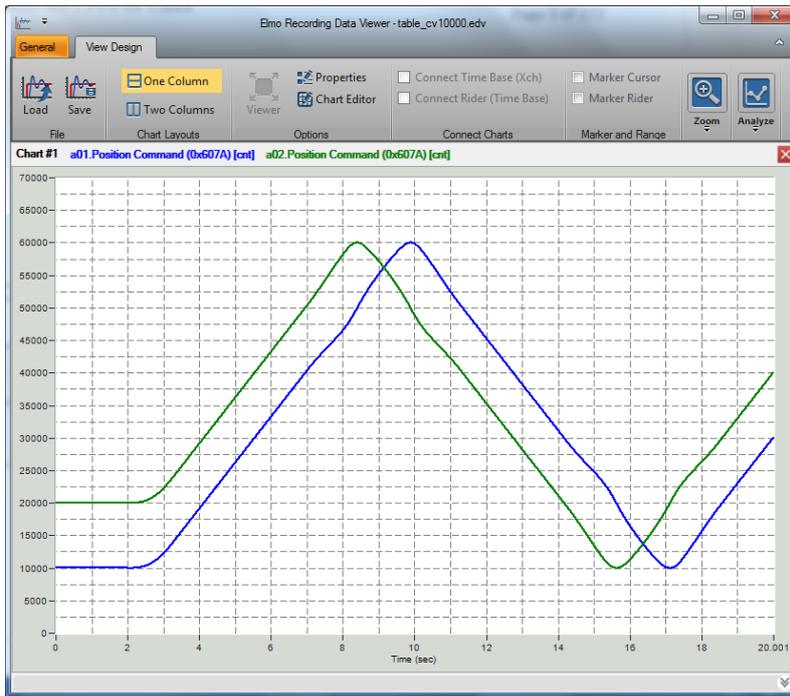
Running the same in FT mode (PT spline type   3   Spline Type = FT) will produce:
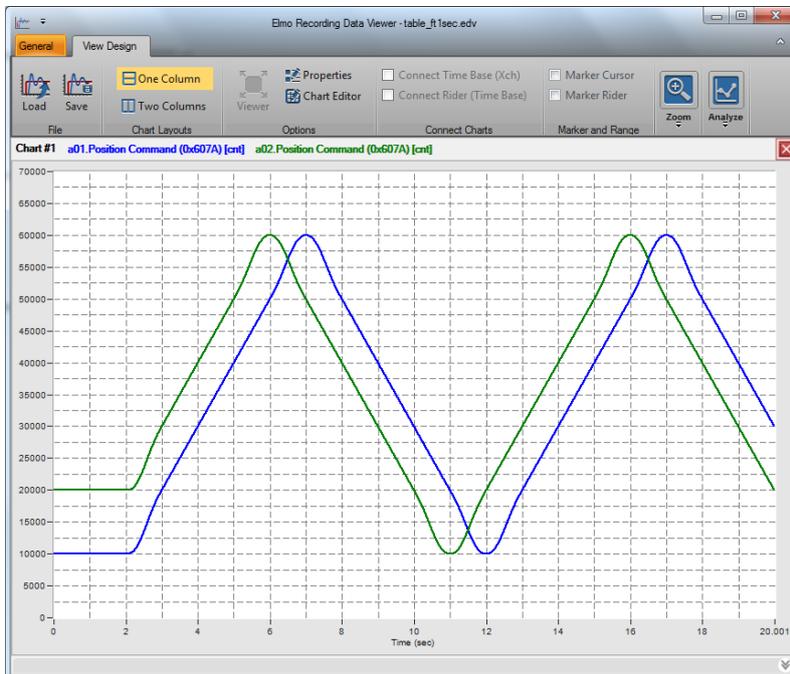
3.2 New ECAM Capabilities

In this new Maesro version, we added 2 new ECAM functionalities:

- Modifying ECAM properties, on the fly.
    - o Ability to modify the CAM slave data, including interpolation types, on the fly.
    - o Ability to change table type – Periodic – to non periodic

This is performed via new API's called:

MMC_CamTableSetCmd

```
typedef  struct _mmc_camtableset_in
{
        double dbTable[NC_ECAM_MAX_ARRAY_SIZE];
        unsigned long ulStartIndex;
        unsigned long ulNumberOfPoints;
        MC_PATH_REF hMemHandle;
        unsigned char ucSpare[32];
} MMC_CAMTABLESET_IN;
```

And:

MMC_CamSetPropertyCmd

Where the input structure is:

```
typedef struct _mmc_camsetprop_in
{
        ECAM_PROPERTIES_ENUM eProperty;
        ECAM_PROPERTY_VALUE value;
        unsigned char ucExecute ;
        unsigned char ucSpare[32]; /**< spare for future use
                                                    */
} MMC_CAMSETPROP_IN;
```

eProperty is currently the periodic mode only.

- New ECAM Interpolation types.
    - o Linear Interpolation
    - o Polynom 5 Interpolation
    - o Polynom 7 Interpolation
    - o Cycloid Position Interpolation
    - o Modified Cycloid Position Interpolation
    - o Sine Trapezoidal Acceleration Interpolation

They are defined as follows:

```
typedef enum {
  eTableDefInterp = 0,
  eLinearInterp = 1,
  ePolynom5Interp = 2,
  ePolynom7Interp = 3,
  eCycloidPositionInterp = 4,
  eCycloidVelocityModified1Interp = 5,
  eCycloidVelocityModified2Interp = 6,
} CURVE_TYPE_ENUM;
```

The newly released EAS – Elmo Aplication Studio, includes a dedicated User Interface for creating, downloading and running ECAM motion profiles:
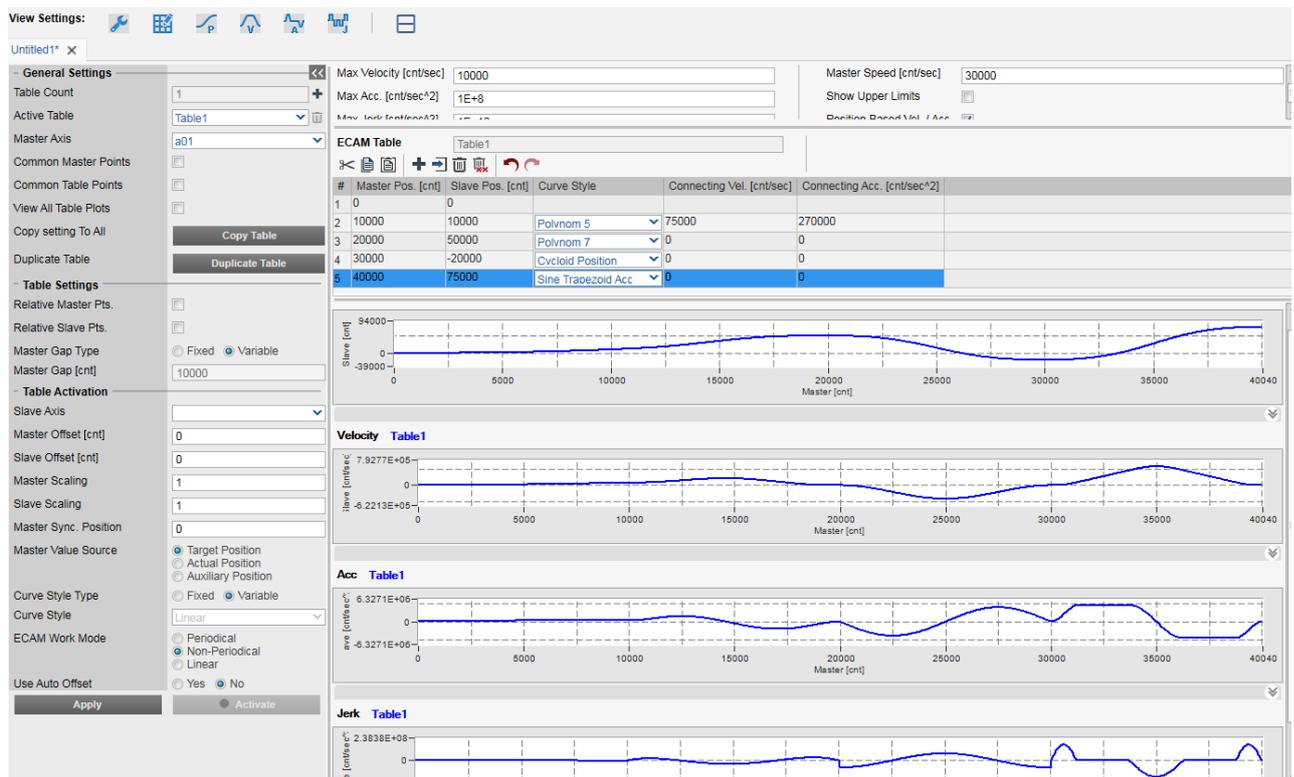
3.3 UVW Rotational Axes Support

Until this version, there was no way to work with Rotational Axes in the Maestro. As part of the SetKinematics API, one may define to work with U, V, W axes.

U, V, W axes are the rotational axes of X, Y, Z – respectively.
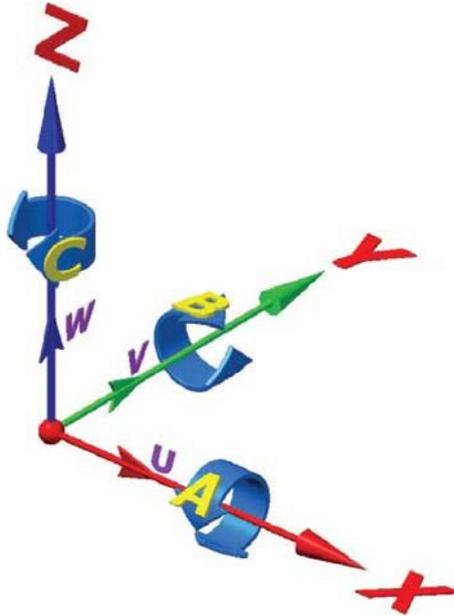
UVW axes behave in a similar manner as the N1, N2... axes do:

- When the Cartesian axes are in motion –the UVW axes start and end together with the Cartesian axes. The User Velocity input is for the Cartesian axes ONLY.

- U,V,W may move separately – without the Cartesian axes. The User velocity in this case is the rotational velocity for U, V, W (In the user units of the U, V, W).

As opposed to N axes, where the N axes must be defined consecutively – here, we can define U, V or W.

Data recording of the U, V, W axes is now available in this version. The EAS signal selector looks as follows:
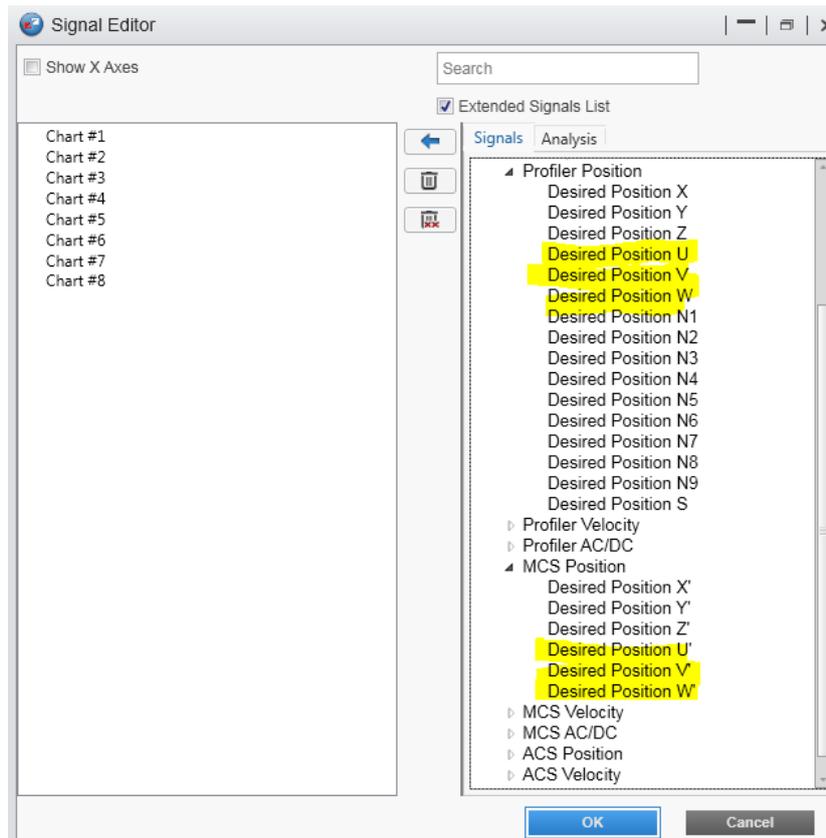
The parameters are of course available for velocity and AC/DC as well.

The order of calling an MCS motion including the new UVW axes is as follows:

The axes kinematics direction is defined in the SetKinematics API, as follows (From EAS script manager screen – Group axis consisting of 4 axes XYUV):

Name		Set Kinematic Transform		
a01.Move Absolute	◢	Group Name	v01	∨
a01.Move Absolute		Axes Count	4	
a01.Move Absolute		Show All Axes	☑	
v01.Set Kinematic Transform	◢	Node		
		[0]	a01	∨
		[1]	a02	∨
		[2]	a03	∨
		[3]	a04	∨
		[4]		∨
		[5]		∨
		[6]		∨
		[7]		∨
		[8]		∨
		[9]		∨
		[10]		∨
		[11]		∨
		[12]		∨
		[13]		∨
		[14]		∨
		[15]		∨
	◢	Type		
		[0]	NC_X_AXIS_TYPE	∨
		[1]	NC_Y_AXIS_TYPE	∨
		[2]	NC_U_AXIS_TYPE	∨
		[3]	NC_V_AXIS_TYPE	∨

3.4 New Supported Kinematics

Scara Robot Kinematics

The new firmware version now supports Scara Robot Kinematics. SCARA acronym stands for **S**elective **C**ompliance **A**ssembly **R**obot.

In general, SCARAs are 4-axis robot arms. They are designed to imitate the action of a human arm and commonly used in pick-in-place, assembly and packaging applications.

(a)   (b)
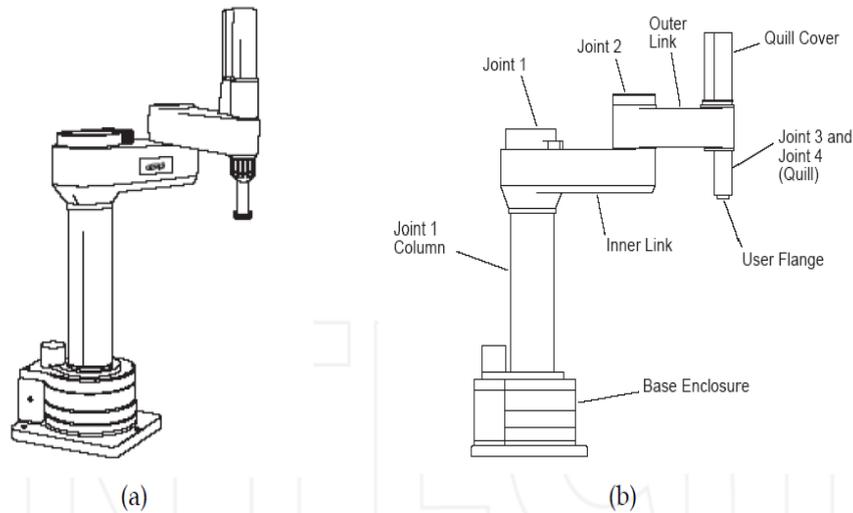
SCARA robot has 4 joints which are linked to the robot. Joint 3 is a translational joint which can move along Z-axis while joints 1, 2 and 4 are rotational joints.

First joint is the base joint and it is also called "the shoulder" as its function looks like a human shoulder. In this joint, the rotational movement of the inner link is provided.

Second joint is called "the elbow" as its function looks like a human elbow. In this joint, the outer link and the inner link are linked. The robot can be programmed to move like a human left or right arm ("Lefty" or "Righty").

(a)   (b)
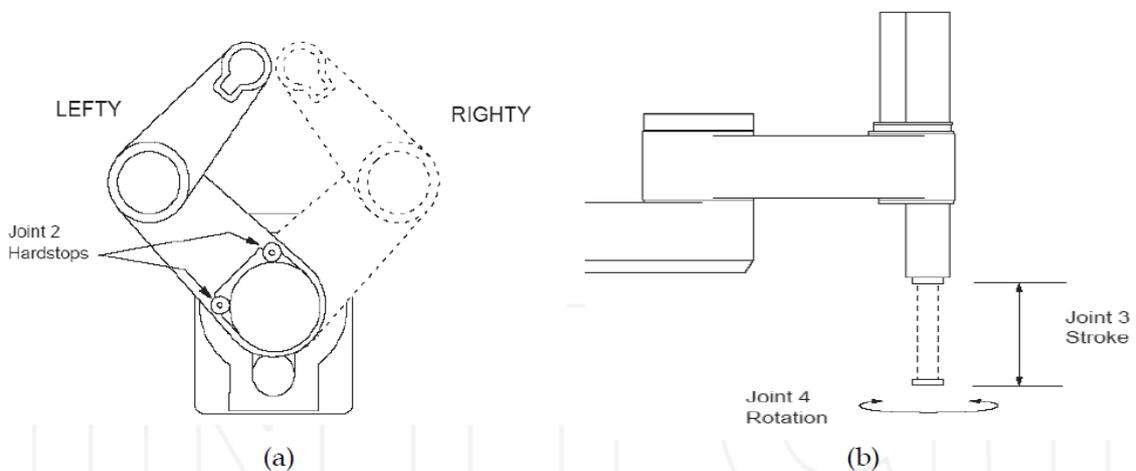
Third joint is a prismatic joint. It is placed at the end of the outer link.

Forth joint is also called "the wrist". Its function is similar to a human wrist and it can be rotated to tighten a bolt or unscrew a screw.

In order to work with all Maestro API's and Scara Robot, the user must call the dedicated API for Scara Robots:

- o MMC_SetKinTransformEx with the NC_SCARA_ROBOT_TYPE as the eKinType

- o MC_SetKinTransformScara – From C++ Interfaces.
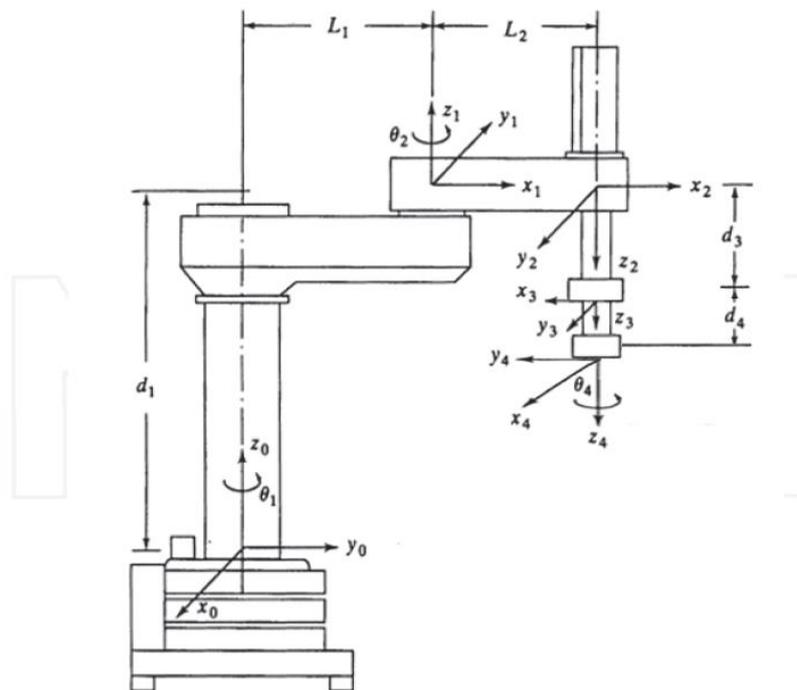
Here are the mechanical parameters which are used for the transformations:

- Inner link length l_1
- Outer link length l_2
- Shoulder offset (column height) d_1
- Offset between axis 3 an 4 in Z direction (wrist offset) d_4
- Elbow sign ES

With the following structure of parameters:

```c
typedef struct
{
    double  dInnerLinkLength;
    double  dOuterLinkLength;
    double  dShoulderOffset;
    double  dWristOffset;
    double  dWristTheta2OffsetCoef;
    MC_KIN_NODE_DEF sNode[NC_MAX_NUM_AXES_IN_NODE];
    int iNumAxes;
    char  cElbowSign;
    char  cPadding1 ;
    char  cPadding2 ;
    char  cPadding3 ;
}MC_KIN_REF_SCARA;
```

Kinematics consists of forward and inverse kinematics.
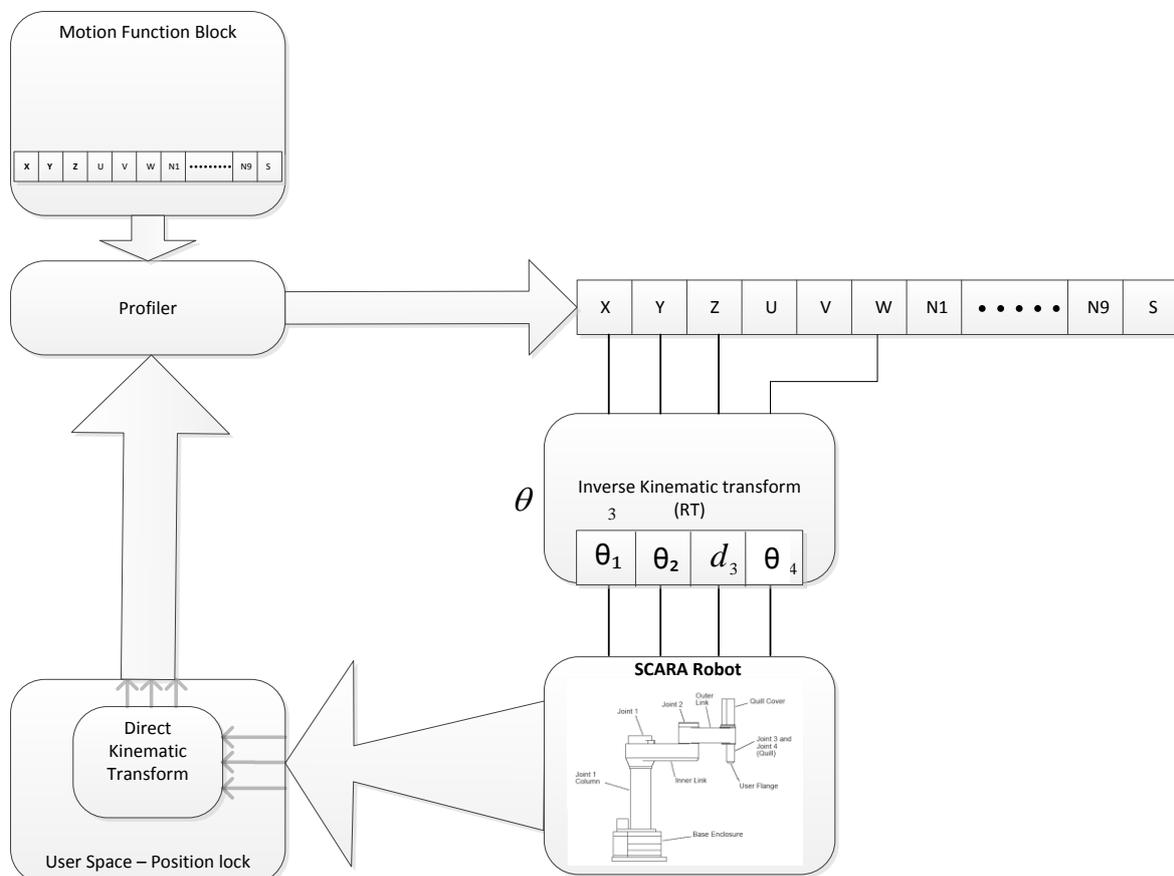
In order to perform a motion from a 'Lefty' robot to a 'Righty' robot – one must go to the 'Zero Robot' position, and call the SetKinematics function again.

3 Link Robot Kinematics

The new firmware version now supports 3 Link Robot Kinematics.

3 Link Robot is a robot that consists of three degree-of-freedom planar robot arm. The arm consists of one fixed link and three movable links that move within the plane. All the links are connected by revolute joints whose joint axes are all perpendicular to the plane of the links.
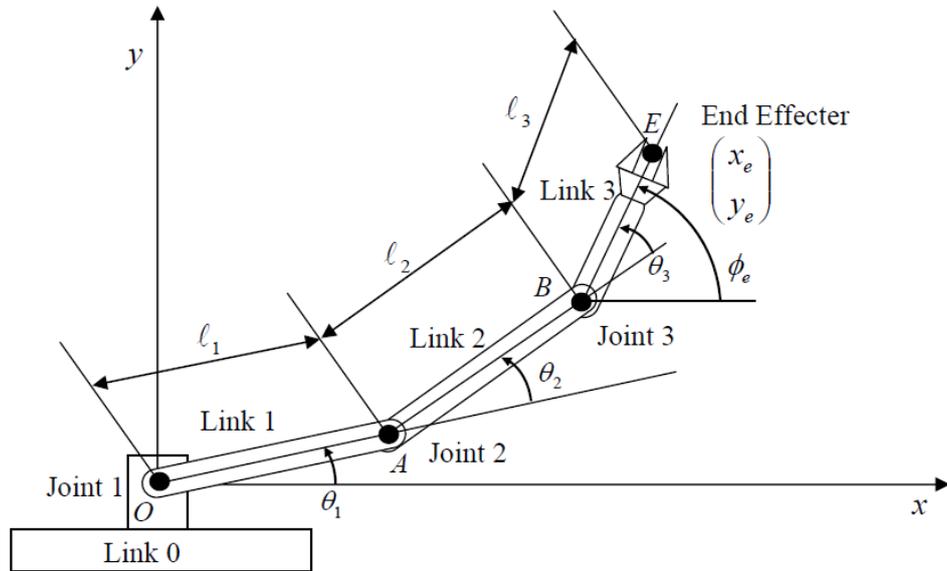
In order to work with all Maestro API's and 3 Link Robots, the user must call the dedicated API for 3 Link Robots:

- o MMC_SetKinTransformEx with the NC_THREE_LINK_ROBOT_TYPE as the eKinType

- o MC_SetKinTransform3Link – From C++ Interfaces

Kinematics Parameters

- o Link Lengths

- o Shoulder and wrist offsets.

- o Left / Right robot

With the following structure of parameters:

```c
typedef struct
{
    double dInnerLinkLength ;
    double dMediumLinkLength;
    double dOuterLinkLength;
    double dShoulderOffset;
    double dWristOffset;
    double  dWristTheta2OffsetCoef;
    MC_KIN_NODE_DEF sNode[NC_MAX_NUM_AXES_IN_NODE];
    int iNumAxes;
    char cElbowSign ;
    char cPadding1 ;
    char cPadding2 ;
    char cPadding3 ;
}MC_KIN_REF_THREE_LINK;
```

3.5 PCS Transformation Support

All Group motion API's receive the CoordSystem parameter (applicable coordinate system: ACS, MCS, PCS). For instance – the MoveLinearAbsolute function below:

Until now - The Maestro supported the following coordinate systems:

ACS – Axes Coordinate System: The system of coordinates related to the physical motors and the single movements caused by the single drives

MCS -Machine Coordinate System: - the system of coordinates that is related to the machine.

We are now introducing

PCS - The coordinate system of the product: The PCS is based on the MCS typically by shifting and rotating the MCS.

In order to summarize:

ACS	Axis related
MCS	Machine related
PCS	Product or Workpiece related
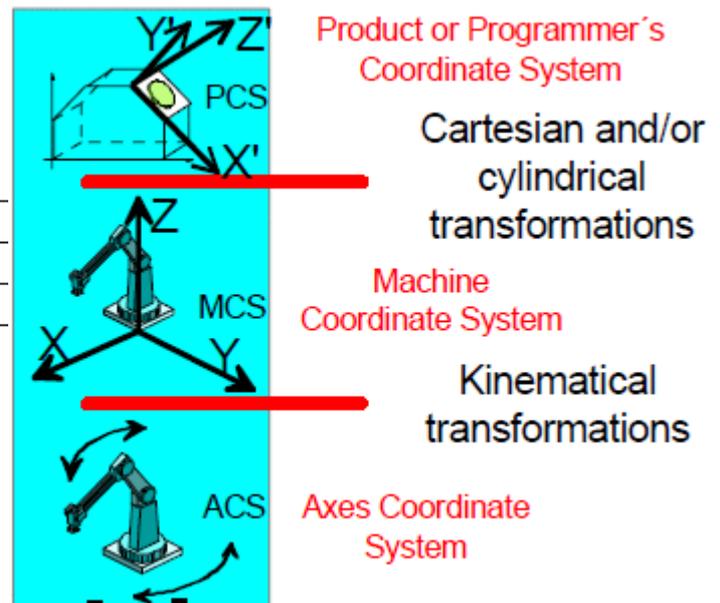
When looking at:
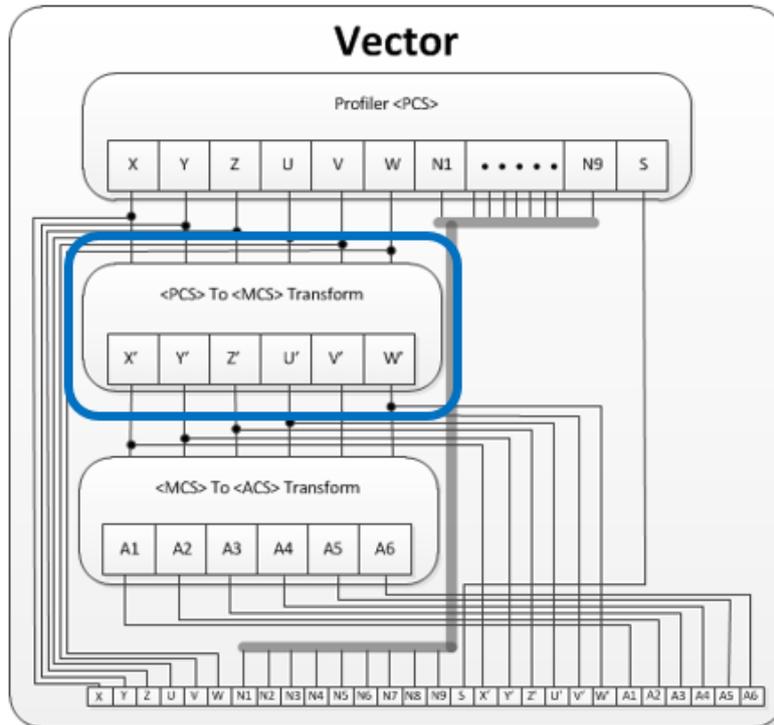
We see that the Maestro profiler works in PCS coordinate system (There was no transformation until now – or transformation was 1:1).

We need to define the <PCS> to <MCS> transformation function.The MC_SetCartesianTransform does exactly this. The User needs to define:

- o 3 x Offsets (XYZ).

- o 3 x Rotations (XYZ).

- o Rotation Units (Degrees / Radians).

Or..:

```
typedef struct
{
     double dOffset[3];       // X,Y,Z translation components
     double dRotAngle[3];     // U,V,W rotation angles
     double dPadding[5] ;
     PCS_ROTATION_ANGLE_UNITS_ENUM eRotAngleUnits;
     MC_BUFFERED_MODE_ENUM eBufferMode;
     MC_EXECUTION_MODE eExecutionMode;
     unsigned char ucExecute;
}MMC_SETCARTESIANTRANSFORM_IN;
```

An example will look as follows:

```
void SetPcs2McsStatic(double dTransX, double dTransY, double dTransZ, double dRotX,
double dRotY, double dRotZ)
{
        MMC_SETCARTESIANTRANSFORM_IN set_transform_in;
        MMC_SETCARTESIANTRANSFORM_OUT set_transform_out;

        set_transform_in.eBufferMode = MC_BUFFERED_MODE;
        set_transform_in.eExecutionMode = eMMC_EXECUTION_MODE_IMMEDIATE;
        set_transform_in.ucExecute = 1;
        set_transform_in.eRotAngleUnits = PCS_DEGREE;
        set_transform_in.dOffset[0] = dTransX;
        set_transform_in.dOffset[1] = dTransY;
        set_transform_in.dOffset[2] = dTransZ;
        set_transform_in.dRotAngle[0] = dRotX;
        set_transform_in.dRotAngle[1] = dRotY;
        set_transform_in.dRotAngle[2] = dRotZ;
        //
        MMC_SetCartesianTransform(g_pHndl,256,&set_transform_in,&set_transform_out);
}
```

This function above shall set the PCS offset and rotation to the values defined in the dTransX, dTransY, dTransZ, dRotX, dRotY, dRotZ.

Once called – All PCS motions will be relative to the 0,0,0 position of the product set in the SetCartesianTransform function.

This function MUST be called for EVERY product/part.

3.6   Track Rotary Table Support

Tracking a Product running on a rotary table is a PLCopen based function.

This function block offers an abstraction layer for a rotary table, assisting the user with tracking objects moving on a circle in space.

The pose of the rotary table relative to MCS is given by a dedicated input of the FB. A further input specifies the initial pose of an object lying on the rotary table. The actual position, velocity, etc. of the rotary table is given by the position of a single axis.
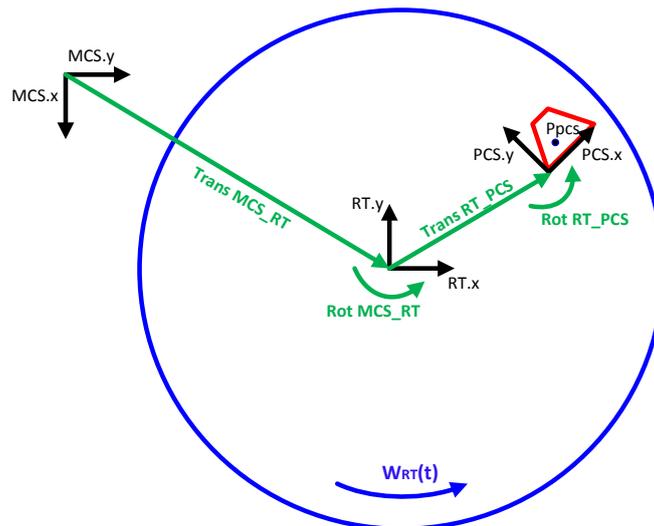
    o   A new API was added:

      MMC_TrackRotaryTable

With the following parameters:

```
typedef struct mmc_trackrotarytable_in
{
      double dRotaryTableOrigin[6];        // X,Y,Z,U,V,W
      double dInitialObjectPosition[6];    // X,Y,Z,U,V,W
      double dInitialRefAxisPosition;
      double dRefAxisScaling;
      double dPadding[5] ;
      unsigned short usRefAxis;
      PCS_REF_AXIS_SRC_ENUM eRefAxisSourceType;       //defines the
                       reference axis source: dTargetPosUU or dActualPosUU
      PCS_ROTATION_ANGLE_UNITS_ENUM eRotAngleUnits;
      MC_BUFFERED_MODE_ENUM eBufferMode;
      MC_COORD_SYSTEM_ENUM eCoordSystem;
      MC_EXECUTION_MODE eExecutionMode;
      unsigned char ucExecute;
}MMC_TRACKROTARYTABLE_IN;
```

The dRotaryTableOrigin parameters are the center of the rotary table in MCS units.

The dInitialObjectPosition array is the distance from the center of the table to the 0,0 defined on the table. In order to move to a defined position on the rotary table, the group axis is to

move to a position, which is PCS based. The PCS position is relative to the 0,0 position given in the `dInitialObjectPosition.`

An example for a Scara robot tracking a rotary table:

```
eKinType = NC_SCARA_ROBOT_TYPE;
eCoordSystem = MC_MCS_COORD;
//
// Intiialize Group axis
v1.InitAxisData("v01",g_pHndl);
//
// Scara axes
a1.InitAxisData("a01",g_pHndl);
a2.InitAxisData("a02",g_pHndl);
a3.InitAxisData("a03",g_pHndl);
a4.InitAxisData("a04",g_pHndl);
//
// Rotary Table
a5.InitAxisData("a05",g_pHndl);
//
// Work in MCS
SetGroupParams(eCoordSystem);


//
// Set SCARA Kinematics
//
// Arm Lengths
set_kin_transex_in.stInput.stScara.dInnerLinkLength = 430;
set_kin_transex_in.stInput.stScara.dOuterLinkLength =370;
set_kin_transex_in.stInput.stScara.dShoulderOffset = 800;
set_kin_transex_in.stInput.stScara.dWristOffset = 200 ;
set_kin_transex_in.stInput.stScara.dWristTheta2OffsetCoef = 4.0 ;
set_kin_transex_in.stInput.stScara.cElbowSign = ElbowSign ;
set_kin_transex_in.eKinType = NC_SCARA_ROBOT_TYPE ;
set_kin_transex_in.stInput.stScara.iNumAxes = 4 ;
//
set_kin_transex_in.eBufferMode = MC_BUFFERED_MODE;
set_kin_transex_in.ucExecute = 1;
//
// Set node axis references:
set_kin_transex_in.stInput.stScara.sNode[0].hNode=0 ; //axis a01
set_kin_transex_in.stInput.stScara.sNode[1].hNode=1 ; //axis a02
set_kin_transex_in.stInput.stScara.sNode[2].hNode=2 ; //axis a03
set_kin_transex_in.stInput.stScara.sNode[3].hNode=3 ; //axis a04
//
set_kin_transex_in.stInput.stScara.sNode[0].eType = NC_ACS_A1_AXIS_TYPE;
set_kin_transex_in.stInput.stScara.sNode[1].eType = NC_ACS_A2_AXIS_TYPE ;
set_kin_transex_in.stInput.stScara.sNode[2].eType = NC_ACS_A3_AXIS_TYPE ;
set_kin_transex_in.stInput.stScara.sNode[3].eType = NC_ACS_A4_AXIS_TYPE ;
//
set_kin_transex_in.stInput.stScara.sNode[0].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC ;
set_kin_transex_in.stInput.stScara.sNode[1].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC ;
set_kin_transex_in.stInput.stScara.sNode[2].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC ;
set_kin_transex_in.stInput.stScara.sNode[3].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC ;
//
// Cts/ Rev on axes. 1024 x Gear of x 100.
set_kin_transex_in.stInput.stScara.sNode[0].ulTrCoef[0]=102400.0 ;
set_kin_transex_in.stInput.stScara.sNode[0].ulTrCoef[1]=1.0 / 102400.0 ;
set_kin_transex_in.stInput.stScara.sNode[0].ulTrCoef[2]=0.0 ;
set_kin_transex_in.stInput.stScara.sNode[1].ulTrCoef[0]=81920.0 ;
```

```
set_kin_transex_in.stInput.stScara.sNode[1].ulTrCoef[1]=1.0 / 81920.0;
set_kin_transex_in.stInput.stScara.sNode[1].ulTrCoef[2]=0.0 ;
set_kin_transex_in.stInput.stScara.sNode[2].ulTrCoef[0]=10000.0 / 111.0 ;
set_kin_transex_in.stInput.stScara.sNode[2].ulTrCoef[1]=111.0 / 10000.0 ;
set_kin_transex_in.stInput.stScara.sNode[2].ulTrCoef[2]=0.0 ;
set_kin_transex_in.stInput.stScara.sNode[3].ulTrCoef[0]=1e-9;//30720.0 ;
set_kin_transex_in.stInput.stScara.sNode[3].ulTrCoef[1]=1e9;//1.0 / 30720.0 ;
set_kin_transex_in.stInput.stScara.sNode[3].ulTrCoef[2]=0.0 ;


//
// Work in MCS
v1.GroupEnable();

v1.GroupReadActualPosition(eCoordSystem,dbActPosition);


dbPosition[0]=795 ;  dbPosition[1]=0 ;    dbPosition[2]=0 ;dbPosition[5]=0;//360 ;
v1.MoveLinearAbsolute(1000,dbPosition,10000,10000,100000);
WaitForStandBy(v1.GetRef());
//
// Move to Init Position on Rotary table. MCS

dbMcsInitPosition[0]=450.0 ;dbMcsInitPosition[1]=450.0 ;dbMcsInitPosition[2]=0;
dbMcsInitPosition[5]=0;//360 ;

dbPosition[0]=dbMcsInitPosition[0]; dbPosition[1]=dbMcsInitPosition[1];
dbPosition[2]=dbMcsInitPosition[2]; dbPosition[5]=dbMcsInitPosition[5];
v1.MoveLinearAbsolute(1000,dbPosition,10000,10000,100000);
WaitForStandBy(v1.GetRef());


//
// Init Tracking Table
set_rotarytable_in.eBufferMode = MC_BUFFERED_MODE;
set_rotarytable_in.eExecutionMode = eMMC_EXECUTION_MODE_IMMEDIATE;
set_rotarytable_in.ucExecute = 1;
set_rotarytable_in.eRotAngleUnits = PCS_DEGREE;
// Set MCS Coordinates of center of table
set_rotarytable_in.dRotaryTableOrigin[0] = 400.0;       //x
set_rotarytable_in.dRotaryTableOrigin[1] = 400.0;       //y
set_rotarytable_in.dRotaryTableOrigin[2] = 0;           //z
set_rotarytable_in.dRotaryTableOrigin[3] = 0;           //u
set_rotarytable_in.dRotaryTableOrigin[4] = 0;           //v
set_rotarytable_in.dRotaryTableOrigin[5] = 0;           //w
set_rotarytable_in.dInitialObjectPosition[0] = 0;//Position relative to center of
object x//dbMcsInitPosition[0] - set_rotarytable_in.dRotaryTableOrigin[0];
set_rotarytable_in.dInitialObjectPosition[1] = 0;//Position relative to center of
object y//dbMcsInitPosition[1] - set_rotarytable_in.dRotaryTableOrigin[1];
set_rotarytable_in.dInitialObjectPosition[2] = 0;//Position relative to center of
object z//dbMcsInitPosition[2] - set_rotarytable_in.dRotaryTableOrigin[2];
set_rotarytable_in.dInitialObjectPosition[3] = 0;//Position relative to center of
object u
set_rotarytable_in.dInitialObjectPosition[4] = 0;//Position relative to center of
object v
set_rotarytable_in.dInitialObjectPosition[5] = 45//Position relative to center of
object w;
set_rotarytable_in.usRefAxis = a5.GetRef();      // axis reference of rotary table
set_rotarytable_in.eRefAxisSourceType = NC_PCS_TARGET_POS;//NC_PCS_ACTUAL_POS;
set_rotarytable_in.dInitialRefAxisPosition = a5.GetActualPosition();
set_rotarytable_in.dRefAxisScaling = 6.28318530717959/65536; //2 PI / 65536 per rev.
in radians
//
MMC_TrackRotaryTable(g_pHndl,v1.GetRef(),&set_rotarytable_in,&set_rotarytable_out);
```

```
//
// Set PCS Coordinates
SetGroupParams(MC_PCS_COORD);


//
// Moving the rotary table, will result in the SCARA robot following the rotary table.


//
// PCS Motions of the SCARA to a part (or position) on the rotary table will result in
// motion that is performed by the scara robot – following the rotary table part.
```

3.7 Data Logging in Maestro

The user program on the Maestro now has logging capabilities. The Maestro, having being Linux based, utilizes the existing syslog service. The following lists the logger capabilities:

- The log file/s may reside in volatile or non-volatile memory
- The log file/s may be uploaded, read and analyzed by a separate software utility
- The logging feature is supported by IPC Maestro programs only:
  - .gexe programs written by the user will have the ability to log.
  - A dedicated class MMCLogger was created in order to support this feature.
  - IEC will have a dedicated FB/s for supporting the logging feature.
- The logging supports severity levels for filtering reasons
- The logging supports a date and time of the log message
- A rotating file mechanism is supported.
- In addition to the local logging – the user has the ability to perform remote logging, via UDP, to a dedicated host IP and port
- The remote logging has a format similar to the local logging format

An example for a remote log (over UDP)will look as follows – For example – using the Kiwi Syslog Console:

Date	Time	Priority	Hostname	Message
04-28-2015	18:08:19	User.Error	192.168.1.3	Jan 15 22:45:47 root: Login
04-28-2015	18:07:59	User.Critical	192.168.1.3	Jan 15 22:45:26 root: System SHUTDOWN
04-28-2015	18:07:42	User.Alert	192.168.1.3	Jan 15 22:45:10 root: System Just Started

A local log:

```
#
# cat LogMsgEg
Jan 14 03:08:08 syslogd started: BusyBox v1.10.0
Jan 14 03:08:21 loggingTst.gexe: Monitor Application: Emergency Movment Stop case #0
Jan 14 03:08:35 loggingTst.gexe: Monitor Application: Not Found ModBus Server #0
Jan 14 03:08:49 loggingTst.gexe: Monitor Application: Not Found ModBus Server #0
Jan 14 03:10:18 syslogd started: BusyBox v1.10.0
Jan 14 03:10:38 loggingTst.gexe: Gmas Tftp Server Started #0
#
#
#
#
```

A dedicated class MMCLogger was created. The API is as follows:

- **Define the Log file path & file-name** — LogMsgConfigureFilePathAndName
- **Remove logging files** — LogMsgConfigureRmoveLogFiles
- **Normal / Smaller logging output** - LogMsgCnfLogMsgSize
- **Set Max File size before rotate** - LogMsgCnfMaxFileSizeB4Rotate
- **Set Number of rotated logs to keep** — LogMsgCnfNumRotateFiles
- **Set Logging to remote** - LogMsgCnfToRemoteHost
- **Activate logging server** - LogMsgStartLoggerServer
- **Stop logging server** — LogMsgStopLoggerServer
- **Configure and start the server (All in 1 function…)**- LogMsgConfigureAndStartLoggerServer
- **Send logging Message to Server** – LogMsg

```
/* Server Configure function:                          */
/* Set Server Configure parameters to defaults.        */
/* !!! Does NOT affect RUNNING SERVER, affects ONLY ON NEXT SERVER START*/
 void    LogMsgConfigureToDefault(void);

/* Server Configure function:                          */
/* Defines Logging file path & name                     */
/* Add extention to Base file name (or user deliver name)    */
/* (The EAS search/find logging files according to this extension).*/
/* If not specify name or its NULL => use the defaults name.   */
/* Return 0 when everything ok and as expecting.         */
/* If not called using the defaults.                 */
 int     LogMsgConfigureFilePathAndName
(char* pcLogMsgFileName = LOG_MSG_DEF_LOG_FILE_NAME,
enLOG_MSG_LOGGING_FILE_LOCATION_PATH enLogMsgLoggingFileLocation =
enLOG_MSG_FLASH_FILE_PATH);

/* Server Configure function:                          */
/* Remove (Delete) specific (or defaults) logging files.    */
/* Add extention to Base file name (or user deliver name)    */
/* (The EAS search/find logging files according this extention).*/
/* If not specify name or its NULL => use the defaults name.   */
/* Return 0 when everything ok and as expecting.         */
/* If not called using the defaults.                 */
 int     LogMsgConfigureRmoveLogFiles(char* pcLogMsgFileName =
LOG_MSG_DEF_LOG_FILE_NAME,
enLOG_MSG_LOGGING_FILE_LOCATION_PATH enLogMsgLoggingFileLocation =
enLOG_MSG_FLASH_FILE_PATH);

 /* Server Configure function:                         */
 /* Normal / Smaller logging context - Set logging message size */
 /* Return 0 when everything ok and as expecting.        */
 /* If not called using the defaults.                */
 int     LogMsgConfigureLogMsgSize(enLOG_MSG_SIZE_ENUM enLogMsgSize =
enLOG_MSG_SIZE_SMALL);
```

```
/* Server Configure function:                                */
/* Define (set) the Max File size before rotate:             */
/* Range: LOG_MSG_MIN_FILE_SIZEKB to LOG_MSG_MAX_FILE_SIZEKB */
/* Default: LOG_MSG_DEF_FILE_SIZEKB                          */
/* Return 0 when everything ok and as expecting.            */
/* If not called using the defaults.                        */
 int    LogMsgConfigureMaxFileSizeB4Rotate(int iFileSizeB4RotateKB =
LOG_MSG_DEF_FILE_SIZEKB);


/* Server Configure function:                                */
/* Set Number of rotated logs file to keep:                 */
/* Range: LOG_MSG_MIN_ROTATE_FILES to LOG_MSG_MAX_ROTATE_FILES */
/* E.g1:  Files for iNumRotateFiles=1 LogMsg, LogMsg.0      */
/* E.g2:  Files for iNumRotateFiles=2 LogMsg, LogMsg.0 LogMsg.1 */
/* E.g3:  LogMsgNumRotateFiles(3);                          */
/*      Cause cyclic write to files:                        */
/*          LogMsg, LogMsg.0, LogMsg.1, LogMsg.2            */
/* Return 0 when everything ok and as expecting.           */
 /* If not called using the defaults.                       */
 int    LogMsgConfigureNumRotateFiles(int iNumRotateFiles =
LOG_MSG_DEF_ROTATE_FILES);


/* Server Configure function:                                */
/* Set Logging to remote (logging also localy when|if explicit */
/* request), define remote IP and Port:                      */
/* If (pcRemoteHostIp == NULL) Configure for NO LOG TO REMOTE */
/* !!! Any Configure take affect only on next servers start !!! */
/* Return 0 when everything ok and as expecting.            */
 int    LogMsgConfigureToRemoteHost(char*   pcRemoteHostIp,
     bool    bReportAlsoLocal = false,
     int     iRemoteHostPort = LOG_MSG_DEF_PORT);


/* Configure and Start server daemon, all parameters should existin call
/* If MsgFileName is NULL => use the defaults name.
*/
/* If pcRemoteHostIp is NULL => no log to remote. */
/* The behavior is like all of related config function are call with the
/* relevant parameters and then calling to start server.
*/
 int    LogMsgConfigureAndStartLoggerServer(char*
pcLogMsgFileName,
enLOG_MSG_LOGGING_FILE_LOCATION_PATH    enLogMsgLoggingFileLocation,
enLOG_MSG_SIZE_ENUM                     enLogMsgSize,
int                                    iFileSizeB4RotateKB,
int                                    iNumRotateFiles,
char*                                  pcRemoteHostIp,
bool                                   bReportAlsoLocal,
int                                    iRemoteHostPort);
```

```
/* Activate Logger Server with configured values (defaults      */
/* or from Configure calls).                                     */
/* This call should start the message collector server (daemon) */
/* which stays running also after the application exit.          */
/* Call to configure methods (above) after Start has no          */
/* effect on the running server!!!                               */
/* Return 0 when everything ok and as expecting.                 */
 int    LogMsgStartLoggerServer();

/* Stop logging server:                                          */
/* This call should stop collecting logging message by server    */
/* (stop the Server daemon) which stays running also after the   */
/* application logout/exit.                                       */
/* Return 0 when everything ok and as expecting.                 */
 int    LogMsgStopLoggerServer();

/* Send logging Message to server:                               */
/* if the server already running (started) the message should    */
/* register in logging file / send to remote or both acording to */
/* server Configure.                                             */
/* Message builds according to format as in printf(forma,...)    */
/* Return 0 when everything ok and as expecting.                 */
/*         E.g:                                                  */
/*         int    iSeq=0;                                        */
/*         …                                                     */
/*         iSeq++;                                               */
/*         LogMsg(LOG_CRIT, "Good Message Number %d", iSeq);    */
/* Call with !!! DEFAULT LEVEL !!!                               */
 int    LogMsg(                 const char* format, ...);
/* ABOVE FUNCTION FOR call with DEFAULT LEVEL !!!                */
/* THIS FUNCTION FOR USER SPECIFIC LEVEL.                         */
 /*
  * Legal values for the SendLevel:
  *     LOG_EMERG LOG_ALERT LOG_CRIT LOG_ERR LOG_WARNING LOG_NOTICE
LOG_INFO LOG_DEBUG
  */
 int    LogMsg(int SendLevel, const char* format, ...);
```

3.8 Beckhoff ADS Protocol Support

Communication between TwinCAT software modules, both located on the same hardware device and on two remote devices, takes place via a standardized and open protocol called

<p align="center">ADS = Automation Device Specification</p>

Each software module belonging to or interfacing directly with TwinCAT has its own ADS Interface. We added the capability in the Maestro to communicate with a Twincat device, over the ADS protocol.
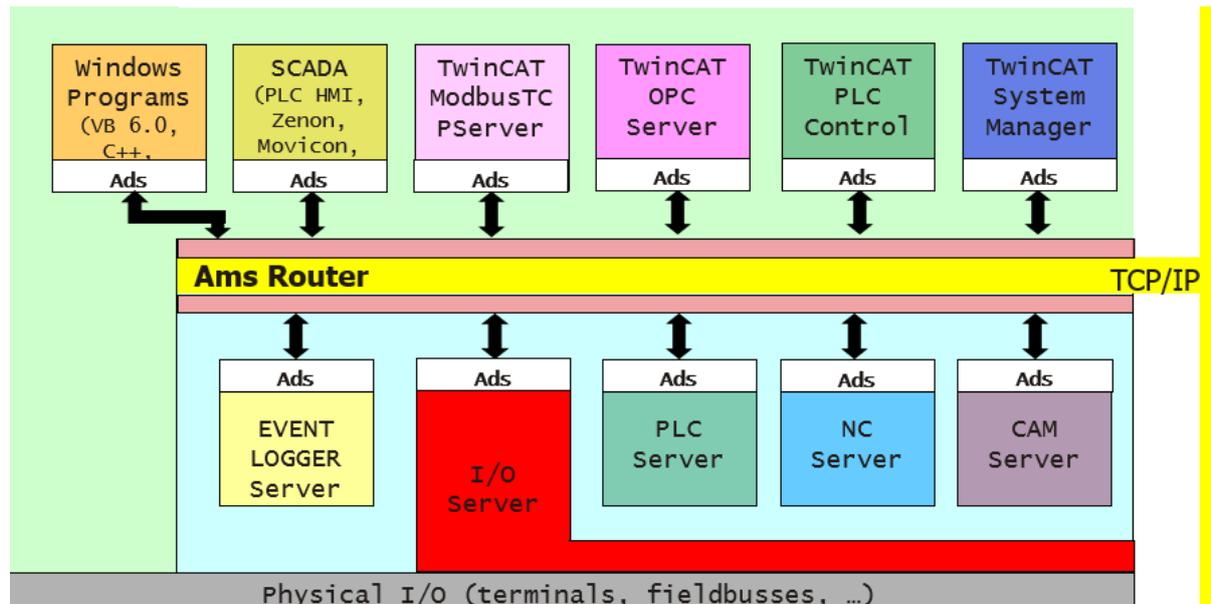
- Maestro acts as an ADS server.

- Reading parameters from the ADS client (usually a Twincat device) consists of defining the parameters within the Twincat program.

- The Maestro supports both UDP and TCP based ADS.

- We added new classes:

    o CMMCAdsAmsTcpProtocol

    o CMMCAdsAmsUdpProtocol

Both including the following methods:

CMMCAdsAmsUdpProtocolInit

CMMCAdsAmsTcpProtocolInit – opens the ports and sets callback functions for the ADS state machines when a Read / Write is called.

3.9   EMBL's - Elmo Motion Blocks Library

As part of this Maestro release, we are introducing Elmo's EMBL library. The purpose of the EMBL is to simplify already existing PLCopen functionalities within the Maestro, by using ready tailored function blocks.

In the near future – these EMBL's will be used as part of project templates that we will introduce.

This library is divided into categories, and each category includes a list of functions. The function documentation can be found in the NetHelp.

EMBL's are available in IEC, C and CPP.

- Single-Axis Homing
    - EM_HomeImmediate
    - EM_HomeOnBlock
    - EM_HomeOnHomeAndLimitSwitch
    - EM_HomeOnIndex
    - EM_HomeOnLimitSwitch
- Multi-Axis Homing
    - EM_MHomeImmediate
    - EM_MHomeOnBlock
    - EM_MHomeOnHomeAndLimitSwitch
    - EM_MHomeOnIndex
    - EM_MHomeOnLimitSwitch
- PVT
    - EM_PVTFile
    - EM_PVTMem
- Joystick
    - EM_JoystickPos
    - EM_JoystickPosAdv
    - EM_JoystickPosExtParam
    - EM_JoystickPosPI
    - EM_JoystickVel
    - EM_JoystickVelAdv
    - EM_JoystickVelExtParam
    - EM_JoystickVelPI
- ECAM
    - EM_CAMMemInit
    - EM_CAMMemActive

- Emulation
  - EM_FbkQuadEmulation
  - EM_FbkPWMSignalEmulation
  - EM_FbkPWMSrcSocketEmulation
  - EM_FbkQuadSignalEmulation
  - EM_FbkWaveSignalEmulation
- Output Compare
  - EM_OCAbsPosInit
  - EM_OCAbsPosActivate
  - EM_OCSnglPlsInit
  - EM_OCSnglPlsActivate
  - EM_OCTimeTabInit
  - EM_OCTimeTabActivate
  - EM_OCPosTabInit
  - EM_OCPosTabActivate
- Position Force
  - EM_POSFORCE
- Group Motion
  - SimpleXYMotion
  - SimpleXYZMotion
- PLCOpen State
  - MultiAxis
    - EM_MPLCOpenIsDisabled
    - EM_MPLCOpenIsError
    - EM_MPLCOpenIsInMotion
    - EM_MPLCOpenIsStandby
  - SingleAxis
    - EM_SPLCOpenIsDisabled
    - EM_SPLCOpenIsError
    - EM_SPLCOpenIsInMotion
    - EM_SPLCOpenIsStandby
- Status Register
  - EM_RegisterIsFLS
  - EM_RegisterIsRLS
  - EM_RegisterIsSWLH
  - EM_RegisterIsSWLL
  - EM_RegisterIsTargetReached
- Wait Functions
  - EM_WaitInMotionMask
  - EM_WaitMotionSettled
  - EM_WaitMotorOn
  - EM_WaitOperationalMode
  - EM_WaitSpecificMask
  - EM_WaitStandStill

- Kinematics
    - EM_Simple2DKinematics
    - EM_Simple3DKinematics

Release Notes - New Firmware Version for GMAS Version 1.1.5.0 and Library Update 260

1. General

The "*uImage_v1.1.5.0.B2_2015_1_27.gms*" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. New Error Handling feature. Ability to define error policies at the system / network and axis level.
2. Hotplug support in the GMAS
    - New mechanism regarding the detection of 'wrong network configuration' at runtime.
    - New validation mechanism.
    - New notification mechanism and behaviour.
3. ECAM support in the GMAS
    - CAM based on a file resided in the GMAS or based on an array.
    - Periodic mode CAM.
    - Non-Periodic mode CAM.
    - Ability to change the CAM during CAM operation.
    - Master/Slave independent scaling and offsetting
    - Absolute and Relative based tables.
    - Auto Offset mode.
    - Fixed gap and non-fixed gap table support.
    - Online modification of an ECAM table.
    - APIs to support the above.
4. New Quintic PVT.
5. New Modulo mode.
6. Send command to drive via SendSDO mechanism.
7. EoE support in IEC programming environment.
8. Send a notification event to the user based on a situation that occurred in the system.
9. Ability to perform data recording on PI variables.
10. Bulk Read on PI Variables. No need to read the variables one by one.
11. WaitForConditionEx – Now the condition can be on a PI variable value as well.
12. WriteGroupOfParametersEx – also for PI variables. Queued and non-queued modes.
13. Ability to acquire the number of PI variables
14. Get PI By User Alias.
15. Ability to receive multiple PI variables information
16. Personality includes description for data recording parameters.

Compatibility Issues

We, as always, do our best in order to maintain compatibility with previously released GMAS versions and libraries. However, in order to work with this G-MAS firmware version the network configuration must be configured via the **EAS Ethercat Configurator – At least from EAS Version 2.1.0.10.**

1.1 Error Handling policies in GMAS

General

Upon the occurrence of an error in the GMAS – the user can define a policy regarding the reaction of the GMAS to the error. The reaction to the error is performed in Real Time.

The policies can be one or more of the following (bitwise selection):

- No reaction
- Stop
- Power Off
- Move System to SAFEOP
- Notification Event to User

The different errors and APIs are described below.

Cyclic Errors (Working Counter), Missed Frames

This error, detected every main cycle – checks for Cyclic PI Read and Write Errors. The error counter must be consecutive – meaning – only when a number of consecutive errors occurred – the reaction/policy will take place.

PHY Errors

The Following Error Counters are monitored:

1. Invalid frame counter of port y
2. RX error counter of port y
3. Forwarded RX error counter of port y
4. ECAT Processing Unit error counter
5. PDI Error counter
6. Lost Link counter of Port y

The value of the PHY errors counter is the maximal (and consecutive) value of all of the above.

AL Error

Once this error is detected, it is reported to the user immediately. The slave will enter the error state upon detection, the user will not be able to change this behavior

The user may define a policy that will be enforced towards the group members of this slave. No policy except ERROR stop will be enforced towards the AL error slave.

Axis Connect / Disconnect / Reconnect

An event will be sent to the user for each of the above occurrences.

1. Upon node disconnection it will enter error state
2. When node reconnects it will stay in the error state until reset by the user.
3. When the first cable (GMAS-> drive in) is disconnected, all drives will enter heartbeat error once and the Ethercat network will be scanned (EEPROMS validation) after reconnection.

Drive Status Word monitoring

The Drive Status Word object is monitored during Real Time. The following are monitored:

o Motor off
o Quick stop reaction
o Fault

An unexpected change in the SW will cause a predefined GMAS policy.

Upon error detection the GMAS will apply the predefined policy on the axis group members, the policy will not be invoked towards the axis that caused the error which will enter ERROR stop immediately.

ERROR	POLICY - Options	Affected axes	ERROR_STOP
MISSED FRAMES		All axes	According to policy

WRONG WC	- Do nothing
	- Perform:
	   ▪ STOP-FB
	   ▪ POWER - FB
	   ▪ SAFEOP
	   ▪ Event		

PHY		- If the detected axis belongs to an active vector – all vector members - Detected axis + all axes - Only detected axis	
AL			YES Only axis causing the error and the group enter error
Axis disconnect		- Axis is immediately entered to error stop - Policy applied on other group members - No effect on entire system - If error does not cause the axis to FAULT or motor off, the policy is invoked towards the Emergency axis	

Unexpected motor-off	- Perform:
	   ▪ STOP-FB
	   ▪ POWER - FB
	   ▪ SAFEOP
	   ▪ Event		

SW FAULT			
Quick Stop			
FB			
EMERGENCY			

New APIs

Name:

MMC_RegErrPolicy

Syntax:

```
int MMC_RegErrPolicy(MMC_CONNECT_HNDL hConn,
MMC_REGERRPOLICY_IN*
     pInParam, MMC_REGERRPOLICY_OUT* pOutParam);
```

Description:
Registers an Error Policy / ies for a specific axis.

Parameters:
The input structure is as follows:

```
typedef struct
{
        NC_POLICY_ENTRY pPolicies[MAX_REG_POLICY];
        unsigned short usAxisRef;
        unsigned char ucNum;

        unsigned char pSpare[64];
} MMC_REGERRPOLICY_IN;
```

`NC_POLICY_ENTRY pPolicies` – an array of errors and policies to be registered
`unsigned short usAxisRef` – Axis ref relevant for axis errors
`unsigned char ucNum` – number of entries in the array to be registered

The NC_POLICY_ENTRY struct is as follows:

```
typedef struct nc_policy_entry
{
 ERRORS eErrType;
 unsigned char ucPolicy;
 unsigned char ucThreshold;
}
NC_POLICY_ENTRY;
```

`ERRORS eErrType` - Enumerator of error that the policy is to be registered on
`unsigned char ucPolicy` bitwise value that defines the policy options:

-   0 – No reaction
-   0x1 – Send notification by event
-   0x2 – all drives perform stop FB
-   0x4 – all drives enter SAFEOP state - Move Axis\System to SAFEOP
-   0x8 – perform EEPROM scan of the entire network
-   0x80 – apply policy to the entire system

The output structure is standard and returns the status and error:

```
typedef struct
{
```

```
        unsigned short usStatus;
        short sErrorID;
    } MMC_REGERRPOLICY_OUT;
```

Code Example：

```
MMC_REGERRPOLICY_IN RegIn;
MMC_REGERRPOLICY_OUT RegOut;
unsigned short usPolicies;
int i = 0;

RegIn.usAxisRef  = usAxis;
RegIn.ucNum      = 3 ;

printf("eNODE_ERROR_ECAT_PHY_ERROR selected x%X\n",eNODE_ERROR_ECAT_PHY_ERROR);
RegIn.pPolicies[0].eErrType = eNODE_ERROR_ECAT_PHY_ERROR;
printf("eSYS_ERROR_MISSED_FRAMES selected 0x%X\n",eSYS_ERROR_MISSED_FRAMES);
RegIn.pPolicies[1].eErrType = eSYS_ERROR_MISSED_FRAMES;
printf("eNODE_ERROR Unexpexted motorOff  selected 0x%X\n",eNODE_ERROR_UNEXPEC_MO_0);
RegIn.pPolicies[2].eErrType = eNODE_ERROR_UNEXPEC_MO_0;

//
// Set Motor Off + Event policy to all errors.
RegIn.pPolicies[0].ucPolicy = ePOLICY_POWER_OFF | ePOLICY_EVENT;
RegIn.pPolicies[1].ucPolicy = ePOLICY_POWER_OFF | ePOLICY_EVENT;
RegIn.pPolicies[2].ucPolicy = ePOLICY_POWER_OFF | ePOLICY_EVENT;


rc = MMC_RegErrPolicy(ConnHndl,&RegIn,&RegOut);
if(rc)
{
        printf("MMC_RegisterErrorPolicy failed, error %d\n",RegOut.sErrorID);
         PrintError(RegOut.sErrorID);
}
```

Name:

MMC_GetErrPolicy

Syntax:

```
int MMC_GetErrPolicy(MMC_CONNECT_HNDL hConn,
MMC_GETERRPOLICY_IN* pInParam, MMC_GETERRPOLICY_OUT*
pOutParam);
```

Description:

Returns an Error Policy for a specific axis.

Parameters:

The input structure is as follows:

```
typedef struct
{
        ERRORS pErrType[MAX_REG_POLICY];
        unsigned short usAxisRef;
        unsigned char ucNum;
}MMC_GETERRPOLICY_IN;
```

ERRORS pErrType – an array of errors that their policy should be returned
unsigned short usAxisRef – axis reference of the required axis (if axis related)
should be returned
unsigned char ucNum – number of policies to be returned

The returned output structure is as follows:

```
typedef struct
{
        NC_GET_POLICY_ENTRY pPolicies[MAX_REG_POLICY];
        unsigned short usStatus;
        short sErrorID;
        unsigned char pSpare[64];
} MMC_GETERRPOLICY_OUT;
```

NC_GET_POLICY_ENTRY pPolicies – an array of returned policies data:

```
typedef struct nc_get_policy_entry
{
        unsigned char ucPolicy;
        unsigned char ucThreshold;
        unsigned char ucCurrentVal;
}
NC_GET_POLICY_ENTRY;
```

unsigned char ucPolicy – current registered policy
unsigned char ucThreshold - current registered threshold
unsigned char ucCurrentVal- current value of the error counter

Code Example：

```
MMC_GETERRPOLICY_IN        GetIn;
MMC_GETERRPOLICY_OUT       GetOut;

GetIn.usAxisRef = usAxis;

GetIn.pErrType[0] = eNODE_ERROR_ECAT_PHY_ERROR;
GetIn.pErrType[1] = eSYS_ERROR_MISSED_FRAMES;
GetIn.pErrType[2] = eNODE_ERROR_UNEXPEC_MO_0;

GetIn.ucNum = 3;

rc = MMC_GetErrPolicy(ConnHndl,&GetIn,&GetOut);
if(rc)
{
        printf("MMC_GetErrPolicy failed error %d\n",GetOut.sErrorID);
         PrintError(GetOut.sErrorID);
         continue;
}

for(i = 0; i < 3; i++)
{
        printf("policy = %hhu, Applies = %hhu, threshold = %hhu, current =
        %hhu\n",(GetOut.pPolicies[i].ucPolicy & (~0x80)),GetOut.pPolicies[i].ucPolicy >>
        0x07,GetOut.pPolicies[i].ucThreshold,GetOut.pPolicies[i].ucCurrentVal);
}
```

Name:

    `MMC_ResetSystem`

Syntax:

```
int MMC_ResetSystem(MMC_CONNECT_HNDL hConn, MMC_RESETSYSTEM_IN*
pInParam, MMC_RESETSYSTEM_OUT* pOutParam);
```

Description:

This API resets the entire system errors, including PHY error counters , cyclic and missed frames errors, in addition it performs EERPROM scan and changes all nodes to INIT and then to OPERATIONAL state and therefore results in motor off of all drives.

In addition this API might reset GMAS fatal error and return it to fully operational status.

Parameters:

No data is needed for the input structure:

```
typedef struct
{
        unsigned char ucDummy;
}MMC_RESETSYSTEM_IN;
```

Result of the operation is returned in a standard output structure:

```
typedef struct
{
        unsigned short usStatus;
        short sErrorID;
} MMC_RESETSYSTEM_OUT;
```

Code Example：

```
 MMC_RESETSYSTEM_IN ResetIn;
MMC_RESETSYSTEM_OUT ResetOut;
MMC_READBOOLPARAMETER_IN ReadIn;
MMC_READBOOLPARAMETER_OUT ReadOut;

ReadIn.eParameterNumber = MMC_GMAS_INIT_STATE;

rc = MMC_GlobalReadBoolParameter(ConnHndl,&ReadIn,&ReadOut);
if(rc)
{
     printf("MMC_GlobalReadBoolParameter failed error %d\n",ReadOut.usErrorID);
     PrintError(ReadOut.usErrorID);
 }
else
{
     printf("GMAS init state is %ld\n",ReadOut.lValue);
}

 printf("resetting system errors\n");

 rc = MMC_ResetSystem(ConnHndl,&ResetIn,&ResetOut);
 if(rc)
 {
     printf("MMC_ResetSystemErrors failed error %d\n",ResetOut.sErrorID);
     PrintError(ResetOut.sErrorID);
 }

 rc = MMC_GlobalReadBoolParameter(ConnHndl,&ReadIn,&ReadOut);
 if(rc)
 {
     printf("MMC_GlobalReadBoolParameter failed error %d\n",ReadOut.usErrorID);
     PrintError(ReadOut.usErrorID);
 }
 else
 {
     printf("now GMAS init state is %ld\n",ReadOut.lValue);
 }
```

```
rc = MMC_GlobalReadBoolParameter(ConnHndl,&ReadIn,&ReadOut);
```

1.2 Hotplug support in the GMAS

General

Upon cable disconnection the behaviour of the GMAS Ethercat master was such that it caused the drive to perform a motor off of all axes currently connected to the network. It did not necessarily need to be a physical disconnection – it could also be caused by a faulty cable.

The feedback the user would receive would be as follows:
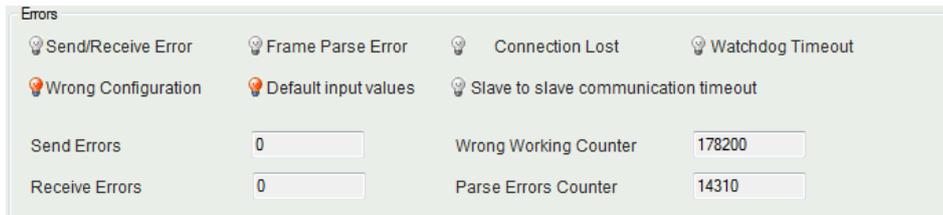
Old Behaviour

A wrong configuration flag would be set (as the number of axes physically on the network does not match the intended configuration). As a result – all the inputs received from the slaves / drives on the network – were set to '0'.

This was the situation until the correct configuration was returned to normal. As a result – the whole network was 'disabled' until initial configuration was returned to normal.

New Behaviour

A new behaviour designed in the GMAS enabled us to properly detect the disconnection and re-connection of slaves on the network without disabling the latter drives on the network. This is done by reading the EEPROM of reconnected slaves ONLY and not all slaves.

'Wrong Configuration Mode' is entered Number of slaves acquired by the *'Working Counter'* is smaller than the number of slaves in the GMAS configuration (slaves disconnected).
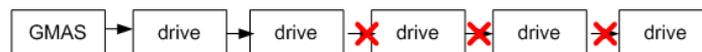
When the GMAS is in the 'Wrong Configuration Mode' – the GMAS seeks for a connection at the disconnection point by reading the slave EEPROM.
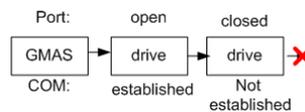
Scanning is continued until receiving the correct FULL configuration as appears in the GMAS configuration file

Network Topologies

Network topologies similar to the following were tested:
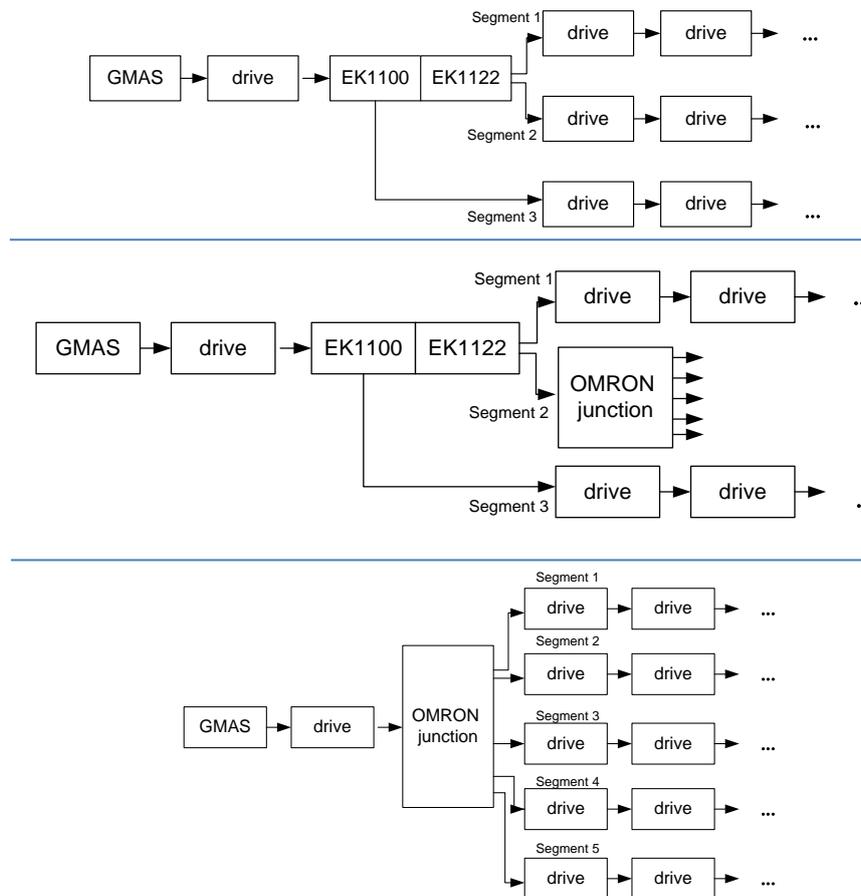
Compatibility issues

The only one who has knowledge of the FULL network topology is the EAS, during the configuration phase. At this stage – the EAS builds the topology with all slaves and junctions in the system. This topology MUST be downloaded to the GMAS, as part of the configuration phase, and is supported in advanced EAS versions ONLY.

Upgrading to this version without re-scanning the system with new EAS will result in *GMAS Fatal Error* at startup.

1.3 ECAM

General

CAM Definition : Cam creates a link between a master and one or more slaves in a position / position mode (see figure hereunder).

With motors and drives one can create the same position / position relationship but in this case via a so called Cam table listing the positions. So the relationship is converted to software and control.
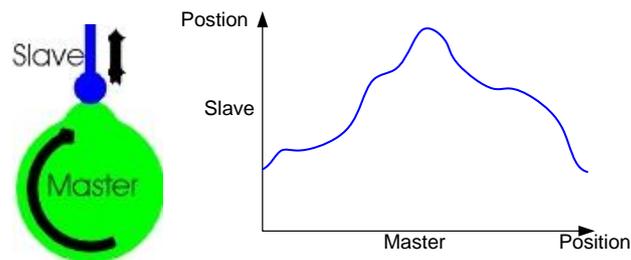
CAM profile illustration

An ECAM example can be found in the graphs below.

Participant axes: master axis a02 (the upper chart), slave axis a03 & slave axis a04 (the lower chart):

CAM process on a03 (blue) runs as a one-shot (non periodic) and a04 as a periodic process. Master a02 of this example moves from 0 to 450000, then stops for a while and moves back to zero. One can see that slave a03 is engaged as long as the master passes along the CAM table and is disengaged when it exits the table for the first time. The other CAM process on a04 proceeds forever until disengaged by MC_CamOut or MC_Stop

ECAM runs as a single axis function block in the system. Several axes can share the ECAM table. The ECAM runs under the fieldbus DS402 Cyclic/Interpolated position motion modes.

The following list specifies the ECAM capabilities.

'One Shot' – Non Periodic Mode

A complete CAM process in a single run along the range of master values. When the master exits the table, the CAM process is aborted automatically.

Cyclic – Periodic Mode

An endless execution of the CAM table. Engagement starts upon ordinary conditions. Master positions are 'so called' modulated and never exit the table range. In order to achieve a smooth curve, another range of slave column is added to the slave calculated position on every master cycle (see examples for MC_CamIn on the follows).

Linear mode - Periodic linear Mode

An endless CAM process. Master /slave engagement starts upon ordinary conditions. Decoupling occurs when master exits the table. CAM process is not completed upon decoupling and proceeds until aborted by MC_Stop (or MC_CamOut). Master may enter and exit CAM table in both directions, so that coupling and decoupling may occur many times as part of this kind of CAM process (see examples for MC_CamIn on the follows).

Bidirectional motion of master axis

Master may move in both directions using any mode of operation (periodic/periodic-linear/one-shot).

Absolute Tables

Absolute table refers to all values as absolute position. Absolute settings of the master/slave are independent of each other

Relative Tables

Relative table refers to all values as relative to master/slave axis position at activation time. Relative settings of the master/slave are independent of each other.

1. It is recommended to use zero values in the first table entry in order to avoid jump.

2. If the first phase in table is not zero then the axis may jump. One can use a slave offset to avoid a jump (negative offset to first phase for instance).

Auto offset

Auto offset is used to avoid any jump of the slave axis. It means that at the engagement stage, slave position in table is adjusted to slave's position. All slave positions in all other rows are adjusted by the same offset.

Auto offset parameter overrides the user definition for slave offset.

Software and Hardware Limits

If the slave axis exceeds the limits during cam operation, an error occurs.

Loading a CAM

CAM tables can be loaded either from an array or from a file residing on the GMAS FLASH.

CAM residing on FLASH

CAM table follows the familiar convention of other tables like PVT, Spline etc… It consists of two parts, header and data. The header (see brown below) defines:

1. Version (optional)
2. Table mode (ECAM file on this case)
3. Dimension: number of slaves. Group is not supported at this phase therefore it must be one for now.
4. Number of points: number of rows within data section.
5. Cyclic: Not in used. It is there for historical reasons.
6.   Position absolute. Not in used. It is there for historical reasons.
7.   Master gap (fixed gap or gap that varies).

Note: Parameters 'Cycle' and 'Position absolute' in header are mandatory for historical reason but the CAM loader doesn't care about them.

Each row is TAB delimited, ended with 'CRLF'. TAB character separates between text and numbers and between numbers in header (see brown below). Text must not contain TAB characters. TAB also separates between columns in the data part. The data part defines the CAM table as lists of master/slave pairs. If master gap defined as fixed (*ECAM fixed gap 1*) then master column is missing. On that case master position relates to 'MasterStartPosition', (set by MC_ CamTableSelect) and calculated by MC_CamIn FB for each segment at runtime.

ECAM version **1** **0**
ECAM mode **4**
ECAM dimension **1**
ECAM num of pts **10**
ECAM cyclic 0
ECAM pos absolute 1
ECAM fixed gap **0**
ECAM data start

```
    0.0000000       0.000000        2
    20000.000000    35000.000000
    50000.000000    60000.000000
    75000.000000    50000.000000
    120000.000000   80000.000000
    144000.000000   110000.000000
    165000.000000   122000.000000
    185000.000000   145000.000000
    200000.000000   160000.000000
    220000.000000   185000.000000 3
```

ECAM data end

Note: If ECAM fixed gap   is 1 then first column, which is the master column, does not appear in

the CAM table.

Loading a CAM Table

Cam loading is done by MC_CamTableSelect API. One may either load CAM table from file or use an array from user program for that matter. In any case a call to MC_CamTableSelect is mandatory. One cannot run MC_CamIn if MC_CamTableSelect was not previously called.

Loading From array

In order to use an array from user program three APIs should be use by the following order:

1. Call MC_CamTableInit to allocate memory in GMAS for a certain CAM table.
2. Call MC_CamTableSelect using the table handler, which was retrieved by the call above to MC_CamTableInit.

Call MC_CamTableAdd, using the table handler above, to load data into table.

Loading From File

Call MC_CamTableSelect with table handler set to (-1) and file path set as appropriate. File format should be compatible to the definitions for CAM table format above.

Flow for loading a CAM

    a.   Load CAM table (from user program or file)

    b.   Call MC_CamIn with input parameters as desired.

    c.   Call MC_CamOut (or MC_Stop) to end the CAM process.

Example:

```
_axes[0].CamTableUnload(-1);              //clear all tables if so desired.
_CamTable.dbGap = 1000;                   //relevant only on fixed gap (see CAM table header)
_CamTable.dbMasterStartPosition = 0;      //relevant only on absolute master and varies gap
_CamTable.eCoordSystem = MC_NONE_COORD;            //for future use
_CamTable.iCamTableID = -1;               //may be retrieved by MC_CamTableInit
_CamTable.pPathToTableFile[0] = "/mnt/jffs/usr/table.cm"; //table path is ignored on memory loading.
_CamTable.eCurveType = eQuinticInterp;    //eQuinticInterp is the default interpolation type.
_CamTable.eTableMode = eCAMT_RWMode;  //modifiable table
_uiTableID = _axes[2].CamTableSelect(_CamTable, 0, 1, 0);//master absolute, slave/relative

uiMaster = _axes[1].GetRef();
_axes[2].CamIn(uiMaster, MC_BUFFERED_MODE, _uiTableID, 0, eCAM_PERIODIC);
While (TRUE)
{
        If (<user program condition>) {
                _axes[2].MC_CamOut();
                 Break;
        }
        usleep(50000);
}
```

1.4 New Interpolation Type in PVT

General

The current version supports **an additional** PVT interpolation type:

"Quintic Interpolation".

Until now – the PVT interpolation supported was of type: "Cubic Interpolation"

The crucial difference between the two interpolation methods is that quintic interpolation guarantees continuity by acceleration while cubic interpolation causes discontinuity by AC at each user defined point.

At the same time the cubic interpolation has a significant advantage: minimal variation by position, velocity and acceleration that is based on the minimal curvature property of the cubic polynomial.

So - the user is advised to make experiments with both methods and choose the one with better results.

Using the new interpolation method

In order to use the new interpolation method – the yellow enums below were added to the NC_MOTION_TABLE_TYPE_ENUM enumeration.

```
typedef enum
{
        eNC_TABLE_NONE,
        eNC_TABLE_SPLINE,
        eNC_TABLE_PVT_FILE,
        eNC_TABLE_PVT_ARRAY,
        eNC_TABLE_PVT_FILE_QUINTIC,
        eNC_TABLE_PVT_ARRAY_QUINTIC_CUB,
        eNC_TABLE_ECAM_FILE,
        eNC_TABLE_ECAM_ARRAY,
        eNC_TABLE_OLSPLN_FILE,
        eNC_TABLE_OLSPLN_ARRAY,
        eNC_TABLE_MAX
} NC_MOTION_TABLE_TYPE_ENUM;
```

They are to be used in the following API's:

- MMC_LoadTableFromFileCmd
- MMC_InitTableCmd
- MMC_AppendPointsToTableCmd

Method Comparison

As mentioned above – users are advised to conduct experiments with both methods in order to choose the best results.

The main difference between the methods is the continuity / discontinuity in the Acceleration.

The graphs below demonstrate the difference in the behaviour of the AC, with the identical table of points.
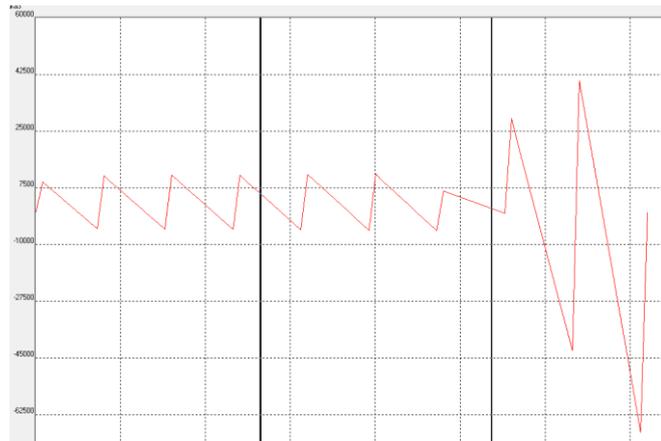
Cubic Interpolation – Discontinuity in AC

The same table, but using the Quintic Interpolation will result in:
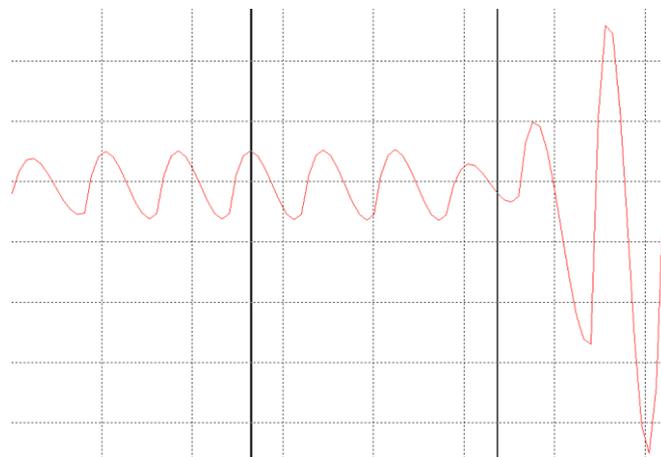
Quintic Interpolation – Continuity in AC

1.5   Additional Modulo Mode in GMAS

There are now 2 Modulo modes that can be set in the GMAS. If we were to look at the EAS interface, we would see the following:

Modulo GMAS Only – The modulo is performed in the GMAS only. The drive lives in the +/- 32 bit position world. The GMAS sends the position to the drive within these limits.

Modulo GMAS and Drive – The modulo exists in the GMAS and the drive. The GMAS will always send the drive values within the Modulo range defined.

The meaning is as follows:

If I have a rotational axis, with 10000 cts/rev. I would like to define user units in degrees:

The user units will be 10000/360= 27.77778

Working in Modulo mode – we will set the limits to be within the 360 degrees:

Modulo GMAS Only

In this mode – the drive is not set to work in Modulo mode:

• Feedback Settings	Modulo Options	No Modulo - Using Position Limits
▶ User Units	Low Position Command VL[3] [cnt]	-9E+8
✔ Display User Units	High Position Command VH[3] [cnt]	9E+8
▶ Limits and Protections	Low Modulo - XM[1] [cnt]	-1E+9
• Current Limits	High Modulo - XM[2] [cnt]	1E+9
⚠ Motion Limits and Modulo		
• Protections		

After performing data recording at the GMAS level, and looking at the Target Position command –

We can see that the Position command exceeds the Modulo value of the GMAS (0-360), and the command is downloaded to the drive in the drive units (cts). The actual position is of course received in both user units (degrees) or drive units (cts), depending on the recorded variable:
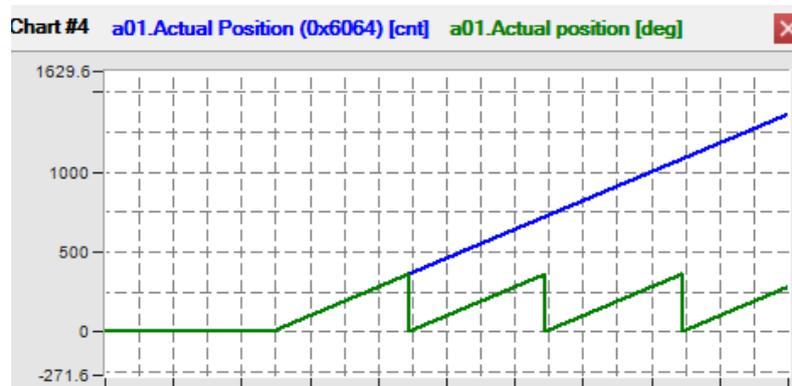
Modulo GMAS and Drive

In this mode, both the GMAS, and the drive, work in Modulo mode. They do not necessarily have to have the same modulo values (GMAs can be in User units, degrees 0-360 and drive can remain in cts)

Setup of Drive Modulo Options:

Modulo Options	Modulo – No Position Limits
Low Position Command VL[3] [cnt]	-1
High Position Command VH[3] [cnt]	10001
Low Modulo - XM[1] [cnt]	0
High Modulo - XM[2] [cnt]	10000

Setup of GMAS modulo options:

Modulo Options	Modulo (GMAS and Drive)
Low Modulo [deg]	0
High Modulo [deg]	360

After performing data recording at the GMAS level, and looking at the Target Position command:
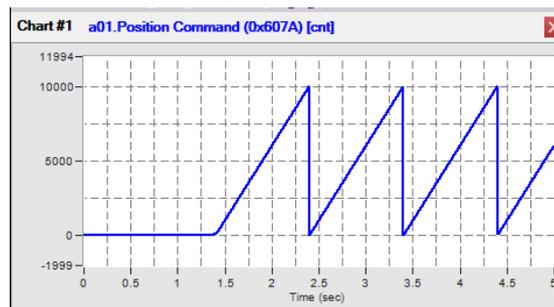
The Position command will be in the modulo range (of the drive). The actual position of the drive and GMAS can be seen below:
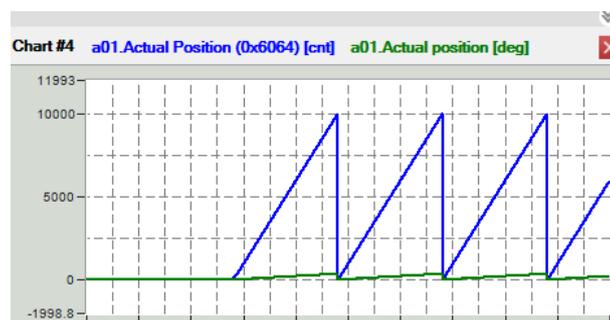
The drive position rolls at drive units(0-10000), and GMAS position rolls at GMAS units (0-360).

1.6 SDO Command To Drive – Using Drive Command Reference

General

The ability to communicate to the drive, in the drive 'language' used to be via EoE (When working over Ethercat) only.

This sometimes caused problems associated with network, masks and gateway definitions.

As the drive supports communication over the SDO mailbox – we simplified the data conversion, and created a group of simple API's under the CPP environment.

This is the preferred method to work in as we experienced factory network issues when working with EoE.

There is no need to define / work with IP addressing when working with this method.

New API's:

```
void SendCmdViaSdoDownload(long   lData, const char* pcCmdIdx, unsigned char ucSubIndex=1)
void SendCmdViaSdoDownload(float  fData, const char* pcCmdIdx, unsigned char ucSubIndex=1)
void SendCmdViaSdoUpload  (long&  lData, const char* pcCmdIdx, unsigned char ucSubIndex=1)
void SendCmdViaSdoUpload  (float& fData, const char* pcCmdIdx, unsigned char ucSubIndex=1)
```

Where:

```
pcCmdIdx – The command to send.

ucSubIndex – Optional Parameter of command index. (Default=1)
```

The data to receive/send may be float or long. The functions are overloaded.

Examples:

```
long glData ;
float gfData ;

// read px
a01.SendCmdViaSdoUpload  (glData, "px");

// read iq
a01.SendCmdViaSdoUpload  (gfData, "iq");


// ui[5]=100
a01.SendCmdViaSdoDownload(100, "ui", 5);
```

1.7 EoE Support in IEC

The IEC now supports EoE to the drive. Users can now communicate with the drives over EoE, and set parameters via this channel

New API's

MC_EoEConnect

```
Inputs:
IpAddr (*STRING*)
Execute (*BOOL*)
Add[] (*USINT*)

Outputs:
sock (*DINT*)
Done (*BOOL*)
Error (*BOOL*)
ErrorID (*INT*)
```

```
              ???
         MC_EoEConnect
  IpAddr               sock
  Execute              Done
  Add[]                Error
                       ErrorID
```

Purpose

Creates an EoE connection to the drive.

Output Values

Sock – Socket ID which is to be used in further API functions

Done – EoE Connect finished OK

Error – If an Error occurred, this bit will be *true*

ErrorID – Error Number.

Example

Usually called at Startup:

fbEoEConnect('10.10.16.16' , true , adder);
dSockt:=fbEoEConnect.sock;

fbEoEConnect1('10.10.16.17' , true , adder1);
dSockt1:=fbEoEConnect1.sock;

MC_EoEClose
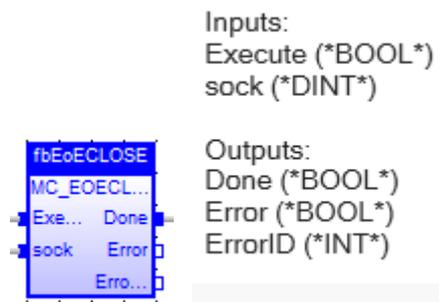
Inputs:
Execute (*BOOL*)
sock (*DINT*)

Outputs:
Done (*BOOL*)
Error (*BOOL*)
ErrorID (*INT*)

Purpose

Closes an EoE connection.

Output Values

Done – EoE Close finished OK

Error – If an Error occurred, this bit will be *true*

ErrorID – Error Number.

Example

MC_EoESetParam

Inputs:
Execute (*BOOL*)
sock (*DINT*)
Add[] (*USINT*)
Command (*STRING*)
@Value (*ANY*)

Outputs:
Done (*BOOL*)
Error (*BOOL*)
ErrorID (*INT*)
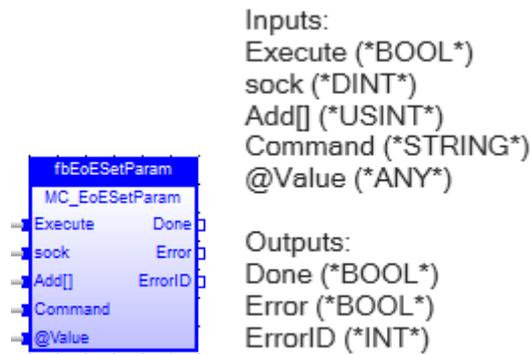
Purpose

Sets a parameter in the drive.

Input Parameters

Execute – Data is sent to drive at rising edge of execute bit.
Sock – Socket returned from the EoEConnect function
Add[] – Address structure consisting of the drive IP address. Usually created with the udpAddrMake function
Value – 'ANY' type parameter.

Output Parameters

Done – Sending the parameter finished successfully.
Error – If an Error occurred, this bit will be *true*
ErrorID – Error Number.

MC_EoEGetParam

Inputs:
Execute (*BOOL*)
sock (*DINT*)
Add[] (*USINT*)
Command (*STRING*)
@Value (*ANY*)

Outputs:
Done (*BOOL*)
Error (*BOOL*)
ErrorID (*INT*)
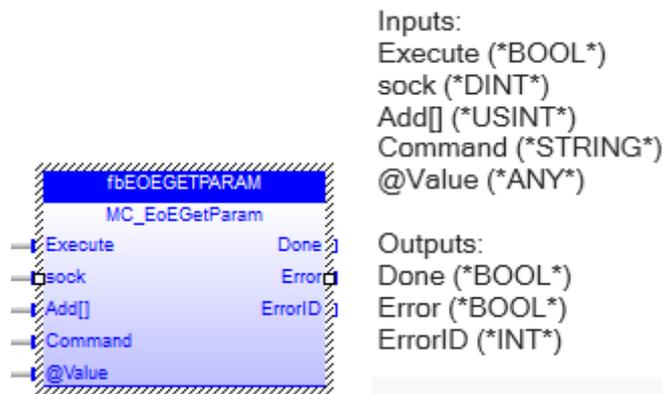
Purpose

Gets a parameter from the drive.

Input Parameters

Execute – Data is sent to drive at rising edge of execute bit.

Sock – Socket returned from the EoEConnect function

Add[] – Address structure consisting of the drive IP address. Usually created with the udpAddrMake function.

Command – Command to send to drive. i.e. 'SP'.

Output Parameters

Done – Sending the parameter finished successfully.

Error – If an Error occurred, this bit will be *true*

ErrorID – Error Number.

Value – 'ANY' type parameter with the value to be sent to the drive.

MC_EoESetArray

Inputs:
Execute (*BOOL*)
sock (*DINT*)
Add[] (*USINT*)
Command (*STRING*)
Index (*UINT*)
@Value (*ANY*)

Outputs:
Done (*BOOL*)
Error (*BOOL*)
ErrorID (*INT*)

Purpose

Sets an array element in the drive.

Input Parameters

Execute – Data is sent to drive at rising edge of execute bit.

Sock – Socket returned from the EoEConnect function

Add[] – Address structure consisting of the drive IP address. Usually created with the udpAddrMake function.

Command – Command to send to drive. i.e. 'UF'.

Value – 'ANY' type parameter with the value to be sent to the drive.

Output Parameters

Done – Sending the parameter finished successfully.

Error – If an Error occurred, this bit will be *true*

ErrorID – Error Number.

MC_EoEGetArray

```
            fbEOEGETARR
            MC_EoEGetArr
  Execute                 Done
  sock                    Error
  Add[]                   ErrorID
  Command
  Index
  @Value
```

Inputs:
Execute (*BOOL*)
sock (*DINT*)
Add[] (*USINT*)
Command (*STRING*)
Index (*UINT*)
@Value (*ANY*)

Outputs:
Done (*BOOL*)
Error (*BOOL*)
ErrorID (*INT*)
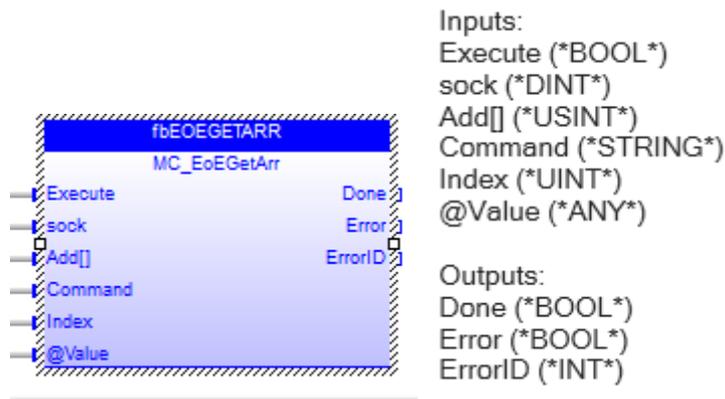
Purpose

Returns an array element in the drive.

Input Parameters

Execute – Data is sent to drive at rising edge of execute bit.

Sock – Socket returned from the EoEConnect function

Add[] – Address structure consisting of the drive IP address. Usually created with the udpAddrMake function.

Command – Command to send to drive. i.e. 'UF'.

Index – Index of array element required.

Output Parameters

Done – Sending the parameter finished successfully.

Error – If an Error occurred, this bit will be *true*

ErrorID – Error Number.

Value – 'ANY' type parameter with the value to be sent to the drive.

1.8   Notification Events

General

The notification events feature allows the user to send a UDP event message based on a situation in the system.

The event is based on the *Administrative queue mechanism* in the system, and can be sent as follows:

- Based on the end of a Function Block.
- Based on a condition in the system.
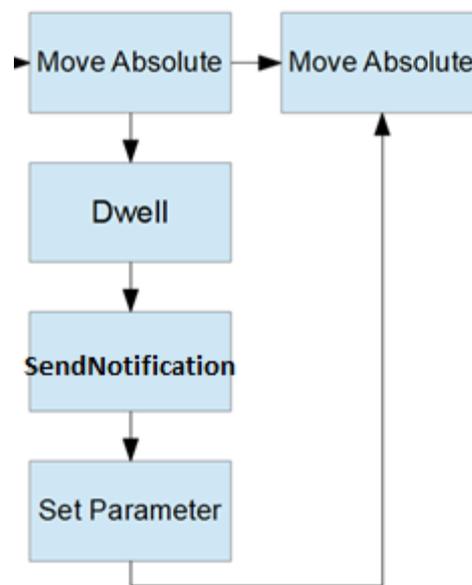
The new SendNotification may include a condition whether to send a notification or not. ONLY if the condition is fulfilled – an event will be sent.

Name

MMC_InsertNotificationFb

Syntax

```
MMC_LIB_API int MMC_InsertNotificationFb(
        IN  MMC_CONNECT_HNDL hConn,
        IN MMC_AXIS_REF_HNDL hAxisRef,
        IN  MMC_INSNOTIFICATIONFB_IN* pInParam,
        OUT MMC_INSNOTIFICATIONFB_OUT* pOutParam)
```

Description:

Inserts a notification event to the Administrative queue.

Parameters:

The input structure is as follows:

```
typedef struct mmc_insnotificationfb_in
{
        int iEventCode;
        unsigned short usNotifyOnCondition;
        unsigned short usSourceAxisRefernce;
        int iParameterID;
        int iParameterIndex;
        double dbReferenceValue;
        MC_CONDITIONFB_OPERATION_TYPE eOperationType;
        long lSpare[2];
}MMC_INSNOTIFICATIONFB_IN;
```

iEventCode – Code to be sent to user when FB is triggered.
usNotifyOnCondition – Flag whether to trigger on condition or not
usSourceAxisRefernce– Source Axis reference
iParameterID   – Parameter ID to check.
iParameterIndex   – Parameter index (if an array).
dbReferenceValue   – Threshold to check against.
eOperationType   – Type of logical operation

Code Example:

```
MMC_INSNOTIFICATIONFB_IN g_stInParams;

printf("NOTIFY ON CONDITION AXIS 1 ACT POSITION (double) LARGER THAN 100\n");

g_stInParams.dbReferenceValue = 1000;
g_stInParams.eOperationType = MC_CONDITIONFB_OP_HIGHER;
g_stInParams.iEventCode = 33;
g_stInParams.iParameterID = MMC_ACT_POSITION_PARAM;
g_stInParams.iParameterIndex = 0;
g_stInParams.usNotifyOnCondition = 1;
g_stInParams.usSourceAxisRefernce = aRef[1].GetRef();
aRef[0].MoveAbsolute(100000,50000,50000,50000,1000000000,MC_BUFFERED_MODE);
aRef[0].InsertNotificationFb(g_stInParams);

printf("Send Event #34 when axis finishes motion\n");

g_stInParams.iEventCode = 34;
g_stInParams.iParameterID = MMC_ACT_POSITION_PARAM;
g_stInParams.iParameterIndex = 0;
g_stInParams.usNotifyOnCondition = 0;
aRef[0].MoveAbsolute(100000,50000,50000,50000,1000000000,MC_BUFFERED_MODE);
aRef[0].InsertNotificationFb(g_stInParams);
```

1.9   Data Recording of PI Variables

Until today the recordable signals are predefined signals that are listed within the personality file.

All the signals are signals which exist in GMAS static memory. In order to add a signal, a new firmware is needed to be created.

The PI variables are allocated dynamically at the GMAS startup according to the configuration file that is downloaded by the EAS, to the GMAS, during the Ethercat configuration process.

The new mechanism enables the user to record every PI variable according to its direction (Input or Output) and variable offset.

The result is – having the following screen within the EAS, that enables us to perform data recording:
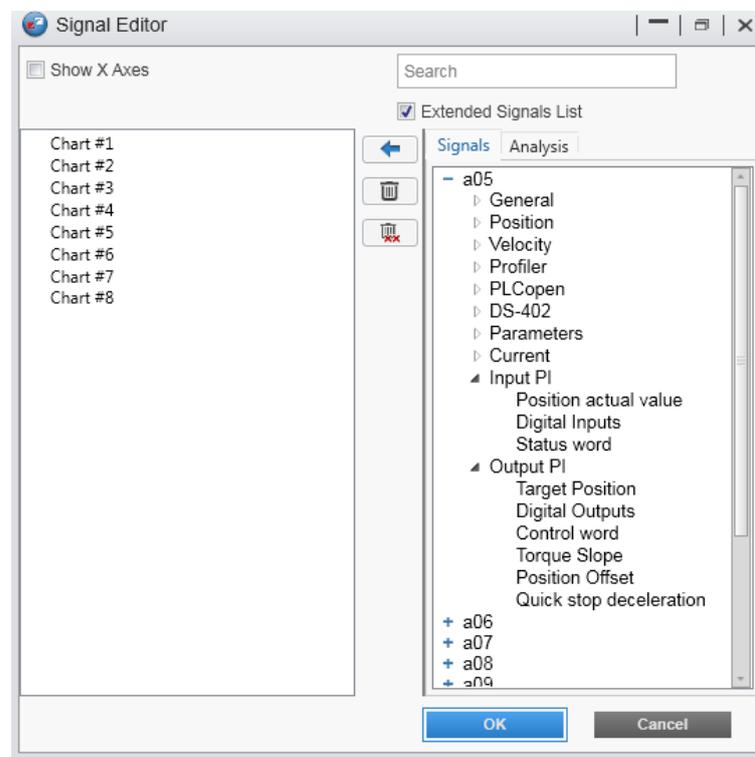
Setting the Recording Variables

After the host (Either EAS or other host software) obtains the PI list in the system:

- By using the Ethercat Configuration currently used in the GMAS
- By calling the GetPIVarInfo or the GetPIVarInfoList API

The host must configure the signal type. A slight change was inserted in the behaviour of the RV array:

uiRv[i] indicates the relevant axis ref (two MSBs) and the signal number (two LSBs). For example:

uiRv[i] = 0x [00][01][00][03] → signal 3 of axis with axis ref 1

We changed the usage of the `uiRv` array. We used the two upper bits of the LSBs of each cell to indicate the PI direction. The upper two bits will hold a `PI_DIRECTION_ENUM`.

The same change will be performed regarding `uiRp[1]` which indicates which variable will be the trigger variable in the same way.

The upper two bits of the LSBs of each cell in `uiRv` and in `uiRp[1]` hold the direction of the PI variable according to the existing numerator changed:

```
typedef enum PI_dir
{
        ePI_INPUT    =  0,
        ePI_OUTPUT   =  1,
        ePI_NONE     =  3,
}PI_DIRECTION_ENUM;
```

In order to define the variable type it is needed to set in the relevant cell in the array (or in the trigger identifier) the type of the variable:

- Regular signals will have to hold from now always *ePI_NONE* in the relevant cell.
- In order to record a *PI input* the relevant cell will be set *ePI_INPUT*.
- In order to record a *PI output* the relevant cell will be set *ePI_OUTPUT*.

When the two upper bits of the LSBs of `uiRv` [i] (or `uiRp[1])` is set to a value that is different than *ePI_NONE* the recorder will consider the lower two bytes of `uiRv` [cell] or `uiRp[1]` (trigger variable number) as PI var offset (not including the direction bits) according to the given direction.

For example:

uiRv[i] = 0x [00][01][80][03] → PI Output number 3 of axis with axis ref 1

uiRv[i] = 0x [00][01][40][03] → PI Input number 3 of axis with axis ref 1

uiRv[i] = 0x [00][01][00][03] → signal 3 of axis with axis ref 1 (same as in the example above)

The rest of the input variables (parameters, signals bit mask, recorder gap and length) are determined in the same manner as they are today. The PI variables size can be acquired from the master xml or from the MMC_GetPIVarInfo API, this size will be used for display purposes.

Since the recorded data is uploaded in 4 bytes blocks, it is necessary that when recording a large PI variable the EAS will also receive the data in consecutive 4 bytes blocks.

Normally when recording a variable of 64 bits, the host software (EAS) marks two consecutive signals for recording. Two bits are set in uiRc and two consecutive signal numbers are set in uiRv.

In order to record a large PI variable it is needed that a bit per 4 byte block will be set in uiRC and the same number of entries in uiRV need to hold the same value (Axis_Ref << 16 | PI var Offset and direction).

Currently we do not support recording of variables larger than 64 bits, therefore two consecutive entries max should be entered in uiRV.

Non Standard Data Types

BOOL

BOOL variables recording are supported as they are often used in IO devices. The BOOL variables are stored in the GMAS RAM according to their offset – up to 8 consecutive BOOL variables are stored in 1 byte. When the user reads a BOOL PI variable, the Read API masks the redundant bits and performs the proper shifting so the user receives an unsigned char with the value of 0 or 1.

In order to spare this processing in RT a whole byte is transferred to the EAS and the EAS performs the masking and shifting according to the bit offset field of the variables for display purposes.

Bitwise and *8 multiple variables* of the size < 64 bits

8_multiple variables are variables of the size 3,5,7 bytes.

Bitwise variables are variables of bit size which is not a whole multiply of 8 – 2..7 bits, 9..15 bits, etc. **Recording of these variables are not supported.**

Variables larger than 64 bits

Recording of these types are not supported.

Triggering PI Variables
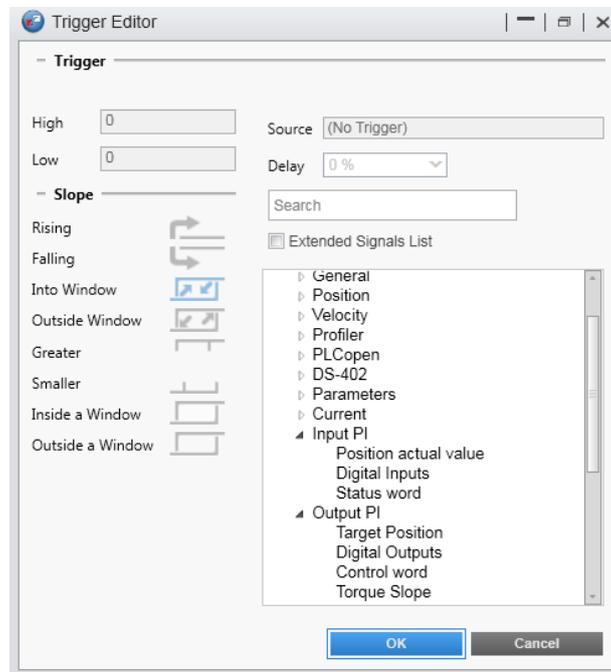
The user is to enter two values of the trigger level in uiRp[4], uiRp[5]. In uiRp[1] the user enters the identifier of the trigger variable : AxisRef << 16 | signal number.

A trigger on an ePI_8MULTIPLE or an ePI_BITWISE variable is not supported.

NOTES:

- o   Using the upper two bits of the LSBs limits the range of PI variables we can record to maximum PI var offset of 16,383 instead of maximum offset of 65,535.

1.10 Bulk Read support for PI variables

General

Since there are no global PI variables, and there is no possibility to create presets for PI variables (there is no predefined name for each variable, therefore we cannot use the existing bulk read mechanism.

There was need to create a new Bulk Read mechanism for PI variables.

Configuring the Bulk Read PI

A new API for configuring the Bulk Read PI was created:

```
int MMC_ConfigureBulkReadPI(
        MMC_CONNECT_HNDL hConn,
        MMC_PICONFIGBULKREAD_IN *pInParam,
        MMC_PICONFIGBULKREAD_OUT *pOutParam)
```

Where, the following structure is the *Input* Structure to the function:

```
typedef struct
{
        PI_BULKREAD_ENTRY pVarsArray[NC_MAX_PI_BULK_READ_VARIABLES];
        NC_BULKREAD_CONFIG_ENUM eConfiguration;
} MMC_PICONFIGBULKREAD_IN;
```

The user is to supply an array of `PI_BULKREAD_ENTRY` in order to read the data later on.

```
typedef struct
{
unsigned short usIndex;
unsigned short usAxisRef;
        PI_DIRECTIONS  eDirection;
} PI_BULKREAD_ENTRY;
```

The user is to supply an array of structures as input:

- usIndex - The PI variable offset
- usAxisRef - The axis ref the signal is related to
- eDirection – Indicates whether this is an input or an output. *ePI_NONE* will indicate the last array entry.

Note:

- PI variables in the same index but in different axes may be different in size (and even do not exist in some axes) depends on the network configuration and the user selections in the configuration stage. Therefore the PI bulk read API will perform validations versus every axis to verify that all the PI variables exist and that the space required for reading all the data is not larger than 1400 bytes. If any error occurs, the user will be notified.
- All variables up to 64 bits are supported except ePI_8Multiple and ePI_BITWISE

Performing Bulk Read

The existing PerformBulkRead() with identical input and outputs structs is used.

1.11 WaitForCondition now supporting PI

General

The WaitForConditionEx FB allows the axis queue to 'wait' for a condition to be fulfilled, before continuing the execution of the rest of the function block. For instance, A MoveAbsolute Function block will not start until the position of a different axis in the system > *value*.
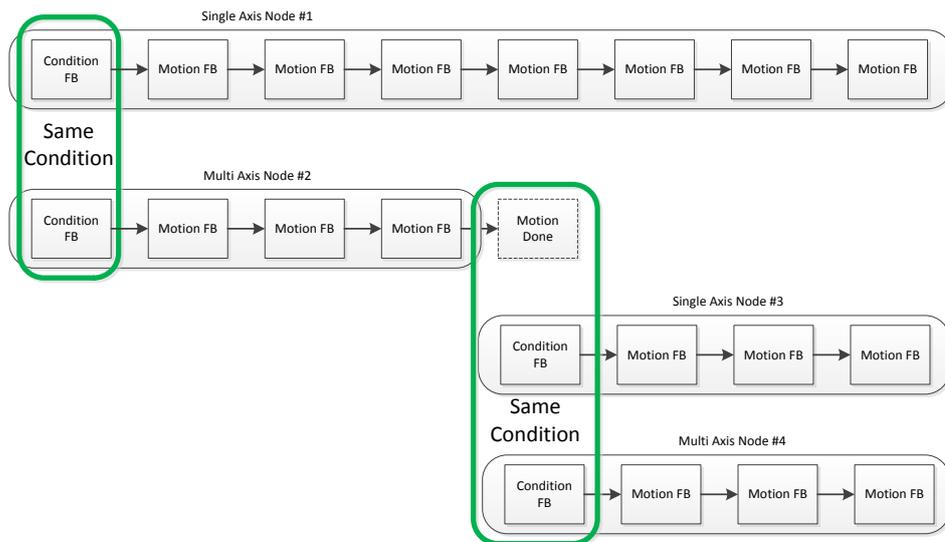
The modification in this version is – the condition to be fulfilled may be on a PI variable. The user will be able to set a condition not only on a parameter from the parameters list, but also on a PI input or output variable. Wait for a PI Input to be set to 1 before continuing.

Syntax

```
MMC_LIB_API int MMC_WaitUntilConditionFBEX(IN MMC_CONNECT_HNDL hConn,
        IN MMC_AXIS_REF_HNDL hAxisRef,
        IN MMC_WAITUNTILCONDITIONFBEX_IN* pInParam,
        OUT MMC_WAITUNTILCONDITIONFBEX_OUT* pOutParam);
```

Description

Waits for a condition to be fulfilled before continuing on the administrative motion/admin queue.

Parameters：

The following structure was added:

```
typedef struct mmc_waituntilconditionfbex_in
{
        double dbReferenceValue;
        int iParameterID;
        int iParameterIndex;
        MC_CONDITIONFB_OPERATION_TYPE eOperationType;
        unsigned long ulDeadlineTimeus;
        unsigned short usSourceAxisReference;
        unsigned char ucExecute;
        unsigned char ucDirection;
        unsigned char ucSpare[20];
}MMC_WAITUNTILCONDITIONFBEX_IN;
```

In order to distinct between a condition on a parameter or a condition on a PI variables the ucDirection field is used. It indicates the PI direction, 3 possible values: ePI_INPUT – PI input var, ePI_OUTPUT – PI output var, ePI_NONE – condition on a parameter.

When ePI_INPUT\ePI_OUTPUT is set to ucDirection we will treat iParameterID as a PI var offset according to the desired direction.

 iParameterIndex – will be ignored

The other fields were used in the same logic (axis ref, operation type and reference val).

Also, there is a possibility to perform WaitUntilConditionEx on variables of type BOOL.

NOTE: Make sure the ucDirection is initialized prior to calling this API.

1.12 WriteGroupOfParameters - now supporting PI

General

The WriteGoupOfParametersEx API now supports the PI interface. This is supported in the *Queued* and *Immediate* modes.

Enhancement

A new API - WriteGoupOfParametersEx API was added. This identical to the WriteGoupOfParameters API – but now also supports writing to PI.

- o Regular parameters will have to hold from now always `ePI_NONE` in the relevant entry `unsigned short usDir`.
- o In order to queue write an output the relevant cell will be set `ePI_OUTPUT`.
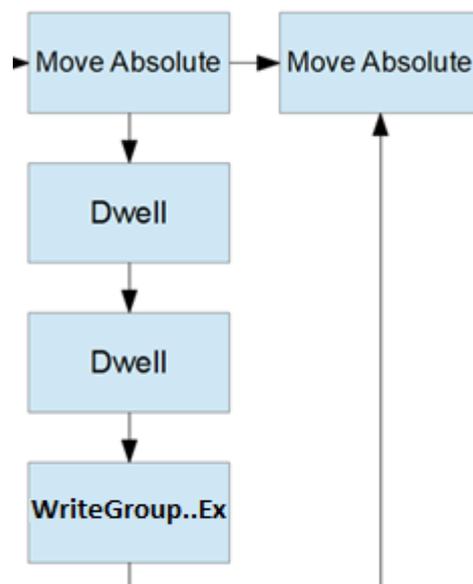
Syntax

```
MMC_LIB_API int MMC_WriteGroupOfParametersEX(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_AXIS_REF_HNDL hAxisRef,
            IN MMC_WRITEGROUPOFPARAMETERSEX_IN* pInParam,
            OUT MMC_WRITEGROUPOFPARAMETERSEX_OUT* pOutParam);
```

Description

Enables the user to write a group of parameters to the administrative queue

Parameters:

The input structure is as follows:

```
typedef struct mmc_writegroupofparametersex_in
{
        MMC_WRITEGROUPOFPARAMETERSMEMBEREX
        sParameters[GROUP_OF_PARAMETERS_MAXIMUM_SIZE];
        MC_EXECUTION_MODE eExecutionMode;
        unsigned char ucNumberOfParameters;
        unsigned char ucMode;
        unsigned char ucExecute;
} MMC_WRITEGROUPOFPARAMETERSEX_IN;
```

Where:

```
typedef struct mmc_writegroupofparametersmemberex
{
        double dbValue;
        MMC_PARAMETER_LIST_ENUM eParameterNumber;
        int iParameterIndex;
        unsigned short usAxisRef;
        unsigned short usPIVarOffset;
        unsigned char ucPiDirection;
        unsigned char pPadding[3];
}MMC_WRITEGROUPOFPARAMETERSMEMBEREX;
```

The usage is the same as the `WriteGroupOfParameters` API. The `ucPiDirection` is to be modified to ePI_OUTPUT in case the user wishes to write to a PI output on the administrative queue.

1.13 Acquiring the number of PI variables

New parameters were added in order to return the number of INPUT and OUTPUT PI variables, per axis.

See:

- `PI_INPUTS`
- `PI_OUTPUTS`

In the parameters explorer.

| PI_INPUTS_NUM | 6 |
| PI_OUTPUTS_NUM | 4 |

Profiler

1.14 GetPI By User Alias

General

Currently the GetPIVarInfo API requires a PI variable offset as an input in order to return a PI var entry structure. We added the functionality in order to use the variable alias as a key.

- This means – the user no longer needs to **'remember'** the offset.
- Adding / modifying the Ethercat configuration may change the offset – but not the

Currently we do not limit the user in the Alias selection. The user may choose an identical name for two PI variables. In addition we need to match between GMAS limited resources to the user need of understandable alias field:

- **Alias string**
    - o (I for inputs or O for outputs).objectNum.Subindex format (same as in the previous PI document). No ability to change. In future, may modify.
- **Display Name**
    - o A new parameter which is to be added to the XML file, and to the relevant EAS windows.
    - o This parameter is not limited in size. It is downloaded as part of the cfg.xml file, but is not parsed by the GMAS.
    - o EAS must check for duplicates and change according to logic above. At scan time, and at download time
    - o This is the parameter that is to be used by the EAS windows. NOT the ALIAS.

For example:

```xml
<slave ID="0">
    <Inputs Size="3">
        <Variable Alias="I0x6064.0" VarType="5" BitSize="32" BitOffs="0" Index="24676" SubIndex="0" VarOffset="0"  DisplayName="Main Actual Position"/>
        <Variable Alias="I0x60FD.0" VarType="5" BitSize="32" BitOffs="32" Index="24829" SubIndex="0" VarOffset="1" DisplayName="Auxiliary Actual Position"/>
        <Variable Alias="I0x6041.0" VarType="4" BitSize="16" BitOffs="64" Index="24641" SubIndex="0" VarOffset="2" DisplayName="Machine feedback Actual Position sensor pressure"/>
    </Inputs>
</slave>
```

New API

The API Input structure:

```c
typedef struct mmc_getpivarinfobyalias_in
{
        char pAliasing[PI_ALIASING_LENGTH];
} MMC_GETPIVARINFOBYALIAS_IN;
```

The user only needs to provide the variable alias.

```
typedef struct mmc_getpivarinfobyalias_out
{
        NC_PI_INFO_BY_ALIAS VarInfo;
        unsigned short usStatus;
        short usErrorID;
} MMC_GETPIVARINFOBYALIAS_OUT;
```

The user receives an output structure similar to the MMC_GETPIVARINFO_OUT structure.

The API is axis related:

```
MMC_GetPIVarInfoByAlias(
                MMC_CONNECT_HNDL hConn,
                MMC_AXIS_REF_HNDL hAxisRef,
                MMC_GETPIVARINFOBYALIAS_IN *pInParam,
                MMC_GETPIVARINFOBYALIAS_OUT *pOutParam)
```

The NC_PI_INFO_BY_ALIAS struct:
```
typedef struct piinfobyalias
{
        unsigned int uiBitSize;
        unsigned int uiBitOffset;
        unsigned short usCanOpenIndex;
        unsigned short usPIVarOffset;
        unsigned char ucCanOpenSubIndex;
        unsigned char ucVarType;
} NC_PI_INFO_BY_ALIAS;
```

API Example

Let's say the user has an output variable called "CW" and he wants to write to it:

```
MMC_GETPIVARINFOBYALIAS_IN GetIn;
MMC_GETPIVARINFOBYALIAS_OUT GetOut;
MMC_WRITEPIVARSHORT_IN WriteIn;
MMC_WRITEPIVARSHORT_OUT WriteOut;

strcpy(GetIn.pAliasing,"CW");
GetIn.ucDirection = ePI_OUTPUT;

// acquire the PI var offset by the alias field
MMC_GetPIVarInfoByAlias(uiConnHndl, usAxisRef, &GetIn, &GetOut);

WriteIn.usIndex = GetOut.VarInfo.usPIVarOffset;
WriteIn.sData = 6;

// write the data to the desired variable
MMC_WritePIVarShort(uiConnHndl,usAxisRef,&WriteIn,&WriteOut);
```

1.15 Receiving Multiple PI Variables

General

Until now, we only had an interface for receiving one PI variable info. On user program startup (or EAS) the user may want to upload large number of PI variables info (or the entire PI inputs\outputs of a slave).

New API

A new API was defined:

```
MMC_LIB_API int MMC_GetPIVarsRangeInfo(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_AXIS_REF_HNDL hAxisRef,
            IN MMC_GETPIVARSRANGEINFO_IN* pInParam,
            OUT MMC_GETPIVARSRANGEINFO_OUT* pOutParam);
```

The user will supply the following input struct:

```
typedef struct mmc_getvarsrangeinfo_in
{
        unsigned short usFirstIndex;
        unsigned short usLastIndex;
        unsigned char ucDirection;
} MMC_GETPIVARSRANGEINFO_IN;
```

And will receive an output struct with the info structs of inputs\outputs on the usFirstIndex to usLastIndex PI var offset according to the desired ucDirection.

The output struct:

```
typedef struct mmc_getvarsrangeinfo_out
{
        NC_PI_ENTRY VarInfo[MAX_PI_VARIABLES_NUM];
        unsigned short usStatus;
        short usErrorID;
} MMC_GETPIVARSRANGEINFO_OUT;
```

The user will be able to receive total of MAX_PI_VARIABLES_NUM = 50 PI variables structs, each PI variable struct:

```
typedef struct pientry
{
        unsigned int uiBitSize;
        unsigned int uiBitOffset;
        unsigned short usCanOpenIndex;
        unsigned char ucCanOpenSubIndex;
        unsigned char ucVarType;
        char pAliasing[PI_ALIASING_LENGTH];
} NC_PI_ENTRY;
```

This struct occupies 28 data bytes. 50 of these structures will occupy 1400 bytes which is the maximal packet size with the addition of the status and error fields.

API Example

In order to upload the entire inputs of a specific slave (no error checking):

```
MMC_READBOOLPARAMETER_IN stInParam;
MMC_READBOOLPARAMETER_OUT stOutParam;
MMC_GETPIVARSRANGEINFO_IN  GetIn;
MMC_GETPIVARSRANGEINFO_OUT GetOut;

stInParam.eParameterNumber = MMC_PI_NUM_OF_INPUTS_PARAM;
stInParam.ucEnable = 1;

// use the proper parameter to acquire the number of inputs (assume less than
the maximum)
MMC_ReadBoolParameter(uiConnHndl, usAxisRef, &stInParam, &stOutParam));

GetIn.usLastIndex = stOutParam.lValue;
GetIn.usFirstIndex = 0;
GetIn.ucDirection = ePI_INPUT;

// get the entire inputs info
MMC_GetPIVarsRangeInfo(uiConnHndl, usAxisRef, &GetIn, &GetOut));
```

GetOut[0] contains the first PI inputs data and the next `ucLastIndex` - 1 cells contain the remaining PI variables data.

If the slave has more than 50 inputs the user may perform several iterations of this operation.

1.16 G-MAS Personality now includes additional information

The data recording section in the GMAS personality now includes 2 extra changes:

- Whether the parameter is an extended parameter – to be viewed in the EAS after setting the checkbox – *Extended Signals List*. These parameters are for the more advanced users.
- A description of the signal. More for the EAS user to display.

Release Notes - New Firmware Version for GMAS

Version 1.1.3.0 and Library Update 252

2. General

The "***uImage_v1.1.3.0.B10_2014_5_28.gms***" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. User Units support
2. Modulus support
3. Offset Support
4. 64 bit counters support
5. MoveTorque API.
6. Administrative Function Block Support
    o ChangeOperationMode.
7. Access to the GMAS Ethercat Process Image
8. Access to the fast reference object in the drive. Support for several working modes.
9. Enhanced Home On Block support
10. Kill Repetitive Support
11. Modbus Client library.
12. User Parameters XML file write support
13. Ability to Start / Stop User applications running on the GMAS.
14. FULL IEC library
15. Group Stop / Stop modification

2.9   User Units Support in the GMAS

Until now, the GMAS did not support User Units. Meaning – the user gave the motion parameters and received the axis positions in the axis units.

In order to work in degrees, or inches – the user had to calculate the ratio into counts, and then perform the motion. Now – it is all performed automatically. The drive remains to work in counts – but when working via the GMAS – the GMAS converts everything to user units.
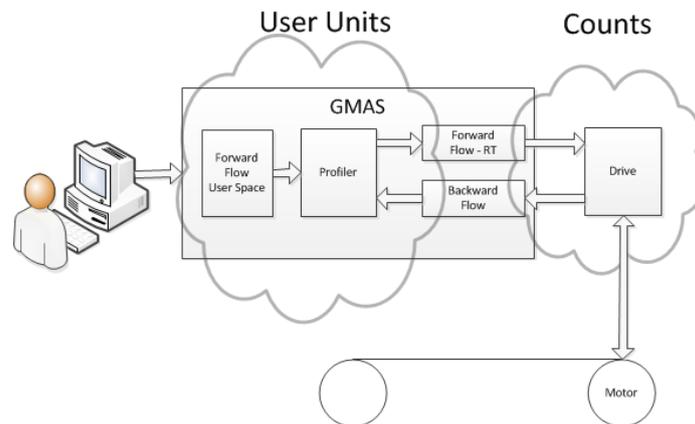
There are 2 different user units the user must set:

- Position UU (User units) – For the Position variables.
- Velocity UU(User Units) – For the Velocity, AC/DC, Jerk variables.

An axis can work in different units for position and velocity – for example – the axis position is defined in degrees, and the speed in RPM:
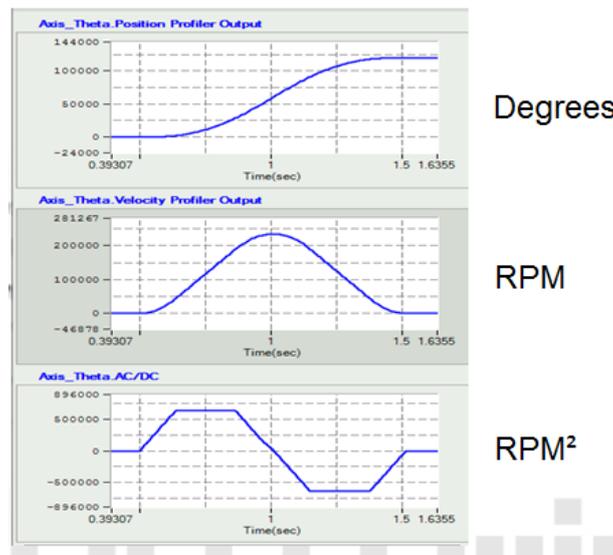
After user units were set – everything is now in user units:

- Motion commands.
- Position readings.

-   Error correction
-   Limit management
-   Modulo
-   Set Position
-   Axis Link
-   Tracking Error

User units is supported in all motion modes:

Single Axis:

NC

NON-NC

Multi Axis:

ACS

MCS

Special Kinematics

Interface

Via the parameters mechanism

MMC_USER_UNITS_POSITION *= 94 – define the position user units ratio.*

MMC_USER_UNITS_VELOCITY *= 95 – define the velocity user units ratio (Vel/AC/DC/Jerk).*

o   The ratio is a double precision floating point variable.
o   The ratio can be a positive number only.
o   The ratio must be greater than 1

Data Recording

The following data recording variables were added:

- Actual position UU
- Actual double position CNT
- Target position UU
- Target double position CNT
- Actual HW position UU
- Actual velocity UU
- Target velocity UU
- Target double velocity CNT
- Desired double ACDC CNT
- Target position UU (total)

For example:

1.9   Modulus Support

In order to support an endless motion and in order to support the CANopen RADO (Rotation Axis Direction option) we introduce the Modulus capability in the GMAS.

General Characteristics of the Modulus capability

- Works in the User Units of the axis.

- Supported for rotational axis only.

- Endless motion of a rotation axis.

- Position Display is within the modulo range.

- Support for asymmetric range.

- Support for all RADO types (Positive, Negative, Shortest, Current Direction).

- Modulo axis cannot have SW limits.

- Can change the *"MMC_MODULO_AXIS"* only in Disabled state.

- Can change the low and high range in any PLC state.

- In *Shortest Way* mode - When the path is the same length in both directions, the axis will move in the negative direction

- ONLY user unit variables are modulated (For instance the Actual Position is not modulated. Only the Actual Position UU is modulated).

Definition Interface

The interface for defining a modulus axis is via the standard parameters mechanism:

- MMC_MODULO_AXIS = 99 – Flag whether the axis is a rotational axis or not.

- MMC_MODULO_LOW = 100 – Defines the low modulo value.

- MMC_MODULO_HIGH = 101 – Defines the high modulo value.

- MMC_ACTUAL_CYCLE = 102 – Counts the number of modulo cycles on axis (actual position).

- MMC_TARGET_CYCLE = 103 – Counts the number of modulo cycles on axis (target position).

Motion Interface

Every Function block has a Direction (or RADO) input:

- Positive.
- Negative.

- Shortest way.
- Current direction

Data Recording

The following recording variables were added:

- Target modulated Position UU
- Actual Position UU (Is already modulated)
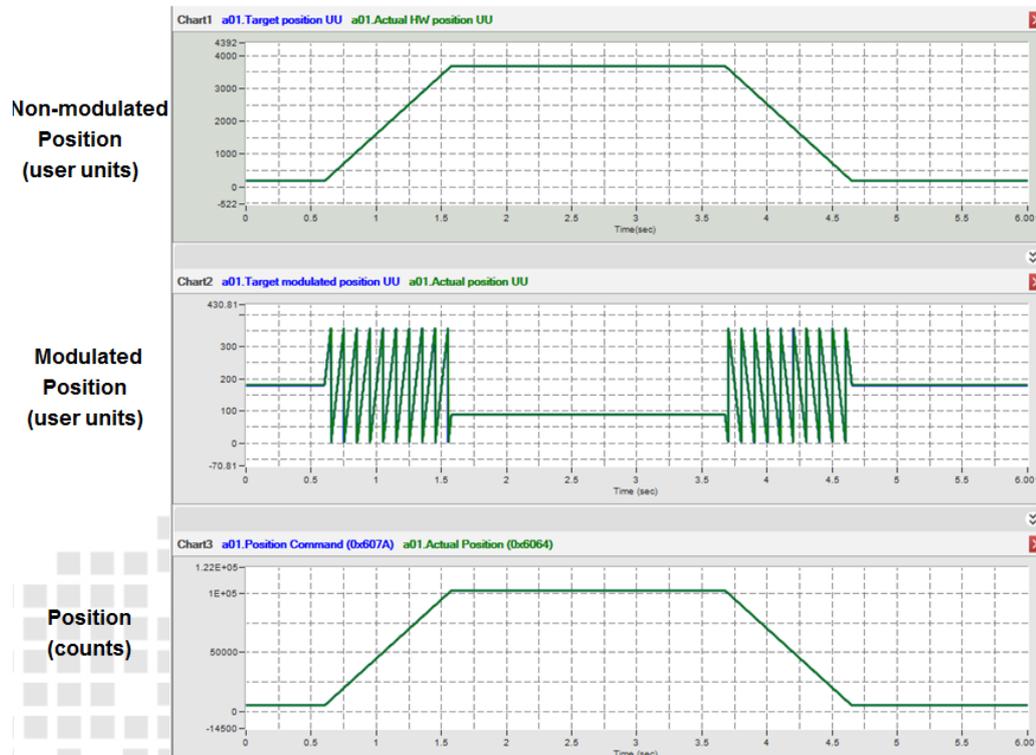
1.10 Position Offset Support

The 'Offset' support in the GMAS allows the user to set a position in the GMAS, without actually changing the position in the physical drive. An offset is held, and is taken into account when motion commands are sent or when position reads are requested.

General Characteristics of the Modulus capability

- Allowed Only in non-moving state.

- All inputs are in User Units
- There are two modes:
    o Absolute mode.
    o Relative mode

Interfaces

- MMC_TARGET_POS_UU = 97 – sets the target position.
- MMC_ACTUAL_POS_UU = 98 – sets the actual position.
  <u>NOTE</u> – The new position can be set only in one of the following states:
    o Disabled
    o StandStill + FB queue is empty

API

- MMC_SetPosition – for one single axis

- MMC_GroupSetPosition – for group of axes

After setting the position 5 x times once axis stopped, a recording will look as follows:
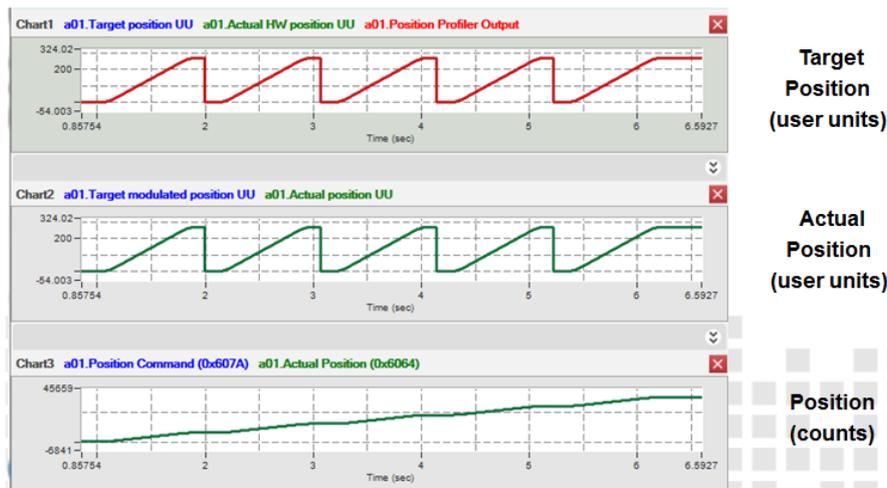
The Position Command to the drive (0x607A) and the Actual Position (0x6064) do not change, as they remain in drive units !

1.11 64 bit counters support

The GMAS until now did not support position commands that were larger than +/- 32bit – whatever the mode (NC or distributed). Although the DS402 communication protocol is limited by 32bit variables, the drive works in an incremental manner and can handle 32bit position rollovers.

The GMAS supports special counters for the 32bit rollovers, and now handles positions in an incremental manner:
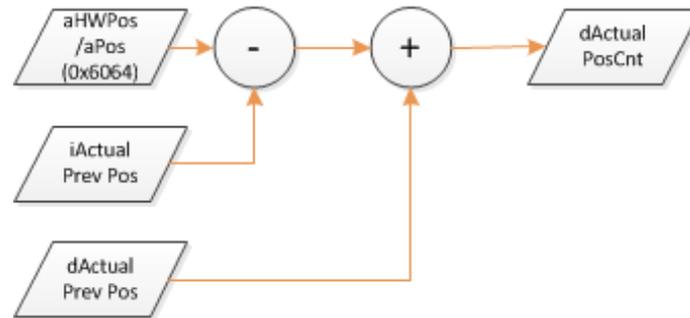
If we were to look at the recording below, we can notice that the 32bit variables 'jump' between the 32 bits:
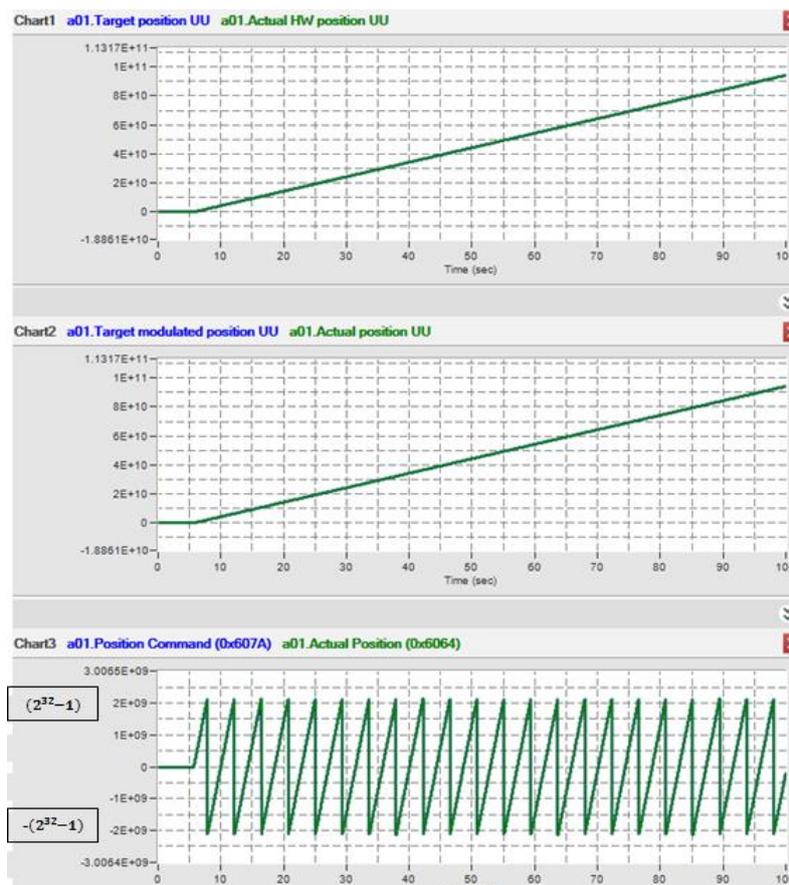
1.12 MoveTorque Function Block

As defined in the DS402 protocol and in PLCOpen standard there are two motion types:

- NC - Distributed – where the master calculates the motion profile and downloads the requested values every cycle.
- Non-Distributed – where each device receives the target point and builds the profile by itself.

When the G-MAS works in NC mode we activate a very complex profiler. The profiler builds a third order s-curve profile that uses high degree of calculations and obtains very smooth motions.

S-Curve profile is a trajectory where the Jerk parameter is a finite value; it means that the Acceleration and deceleration are changed by a given slope.
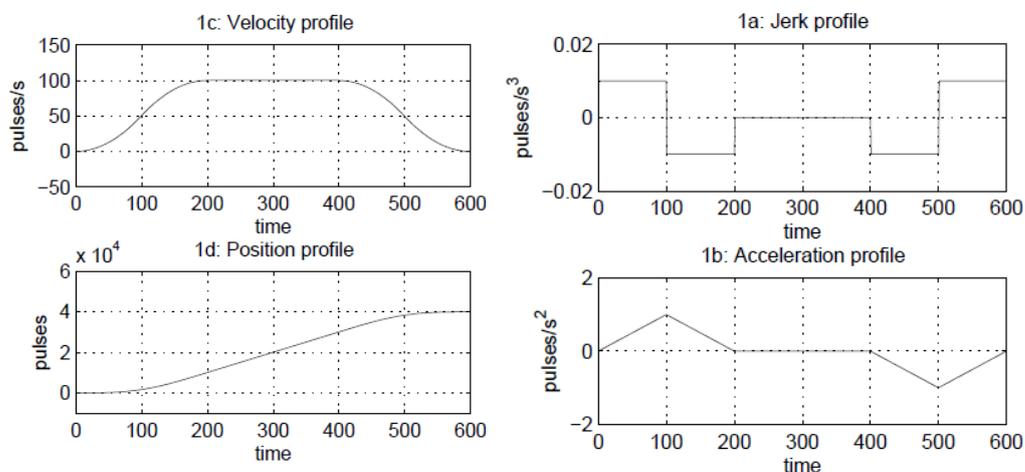
When the G-MAS works in Cyclic Torque mode the user want to build a simple torque profile that will be sufficient to achieve the requested performance.

The simple profile is a trapezoid second order trajectory. In simple profile the jerk value is infinite and the AC/DC is changed immediately to MAX/MIN value.

The simple profiler is more efficient than the s-curve profiler but has the following disadvantages:

- Lack of smoothness because the AC\DC is not continuous.
- Each motion should end at zero velocity – no blending of motions.

In order to get the best performance and usability, we provide the user a new FB MoveTorque that performs the following:

- Uses a new *SimpleProfiler*
- Works in Current Units – User gives the profile values in Mamps , Mamps/sec and Maps/sec^2.
- Can Create a Buffered sequence that contains combination of Cyclic Position movement and Cyclic Torque profile
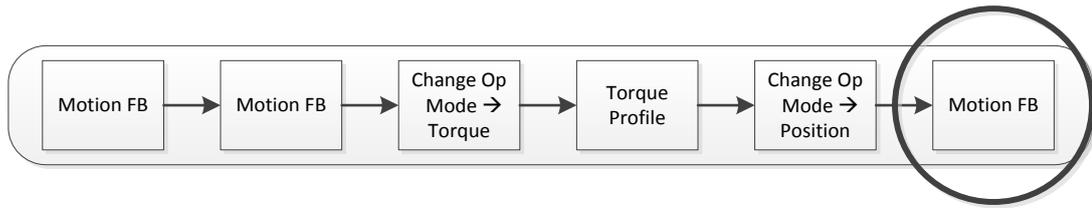
General Characteristics / Restrictions of the Cyclic Torque – MoveTorque Function Block

1. When the axis is in Cyclic torque mode, only MoveTorque can build profiles, all other motion FB's will be blocked.
2. When the axis is not in Cyclic torque mode MoveTorque FB is blocked.
3. MoveTorque FB can be inserted only when the axis is not in Error and not in Disabled state.
4. MoveTorque FB can be inserted only when the axis is not part of an active group.
5. MoveTorque FB is supported in two buffered modes:
    a. Buffered
    b. Abortion
6. In Buffered mode – every motion is finished with velocity = 0
7. In Abortion mode – the current motion is terminated immediately (even when the torque derivative is not 0) and the axis performs the new torque value as fast as possible
8. The user set the target torque in [mA] units.
9. The torque range limited by 16bits variable (0X6071).
10. The ratio between Rated current and UU is set automatically by the G-MAS using the (0cx6075) object.
11. In order to define a torque profile the FB receives the following parameters:
    a. Target torque – positive or negative value.
    b. First derivative of torque change (velocity of torque change) – positive value.
    c. Second derivative of torque change (acceleration of torque change) – positive value.
12. All kinematic inputs will be in same UU.
13. When the axis receives a stop command, the profiler will move to torque "0" value.
14. When the axis receive a halt command, the profiler will move to torque "0" value.
15. When the axis enters the error state the torque value will be immediately set to "0" value.
16. The SW torque limit is checked when inserting a new FB.
17. When changing to torque mode using old API "ChangeMotionMode" – the start value is "0"
18. When changing to torque mode using new API "ChangeOpModeFB" – the start value is defined by the user.
19. The AC/DC/ConstVel bits at the axis level are handled as if it were performed in Position profiler.

API

The function declaration is:

```
/////////////////////////////////////////////////////////////////////////////
/// \fn int MMC_MoveTorqueFB(
///                     IN MMC_CONNECT_HNDL hConn,
///                     IN MMC_AXIS_REF_HNDL hAxisRef,
///                     IN MMC_MOVETORQUEFB_IN* pInParam,
///                     OUT MMC_MOVETORQUEFB_OUT* pOutParam)
/// \brief This function send Move torque command to MMC server for specific Axis.
/// \param  hConn - [IN] Connection handle
/// \param  hAxisRef - [IN] Axis Reference handle
/// \param  pInParam - [IN] Pointer to Move Torque input parameters
/// \param  pOutParam - [OUT] Pointer to Move Torque output parameters
/// \return       return - 0 if success
///                          error_id in case of error
/////////////////////////////////////////////////////////////////////////////
MMC_LIB_API int MMC_MoveTorqueFB(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_AXIS_REF_HNDL hAxisRef,
            IN MMC_MOVETORQUEFB_IN* pInParam,
            OUT MMC_MOVETORQUEFB_OUT* pOutParam);
```

This is the definition of input and output structure:

```
/////////////////////////////////////////////////////////////////////////////
/// \struct MMC_MOVETORQUE_IN
/// \brief Move Torque Command input data structure.
///
/// (See : "Technical Specification
///        PLCopen - Technical Committee 2 – Task Force
///        Function blocks for motion control")
/////////////////////////////////////////////////////////////////////////////
typedef struct
{
    double dbTargetTorque;              ///< Desired Target Torque value
    double dbTargetVelocity;            ///< Maximum Target velocity value
    double dbTargetAcceleration;        ///< Maximum Target acceleration value
    MC_BUFFERED_MODE_ENUM eBufferMode;  ///< MC_BufferMode
    unsigned char ucExecute;           ///< Execution bit
} MMC_MOVETORQUE_IN;

/////////////////////////////////////////////////////////////////////////////
/// \struct MMC_MOVETORQUE_OUT
/// \brief Move Torque Command output data structure.
/////////////////////////////////////////////////////////////////////////////
typedef struct
{
    unsigned int   uiHndl;      ///< Returned function block handle.
    unsigned short usStatus;    ///< Returned command status.
    short  usErrorID;           ///< Returned command error ID.
} MMC_MOVETORQUE_OUT;
```

The MoveTorque FB is implemented in the CMMCSingleAxis class.

```
    void MoveTorque(double dbTargetTorque,
                    MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE)
                    throw (CMMCException);


    void MoveTorque(double dbTargetTorque,
                    double dbTorqeVelocity,
                    MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE)
                    throw (CMMCException);


    void MoveTorque(double dbTargetTorque,
                    double dbTorqeVelocity,
                    double dbTorqueAcceleration,
                    MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE)
                    throw (CMMCException);
```

Parameters

- Maximum torque – mA (symmetric variable)
- Maximum torque velocity
- Maximum torque acceleration

These parameters can be accessed using the regular parameters mechanism.

Data Recording

- Target torque "0x6071" (short variable in rated current).
- Target torque (double variable in UU).
- Target torque velocity (double variable in UU).
- Target torque acceleration (double variable in UU)

1.13 New ChangeOperationMode

Until this version, the operation mode (0x6060 object) was sent via the SDO mechanism. In time crucial applications this caused problems. SDOs are sent as mailbox packets over the Ethercat network. It takes at least 2 mailbox cycles to receive a reply, not to mention the GMAS overhead of updating variables between cycles.

A new 'C' API was introduced in this version:

MMC_ChangeOpModeEx()

This operational mode change occurs in the GMAS Realtime, and not in the background. The mechanism the operational mode is changed is different, as the 6060 must be a mapped variable, and is sent part of the Process Data.

The user can define a motion sequence and push all of the motion function blocks into a queue, such as:
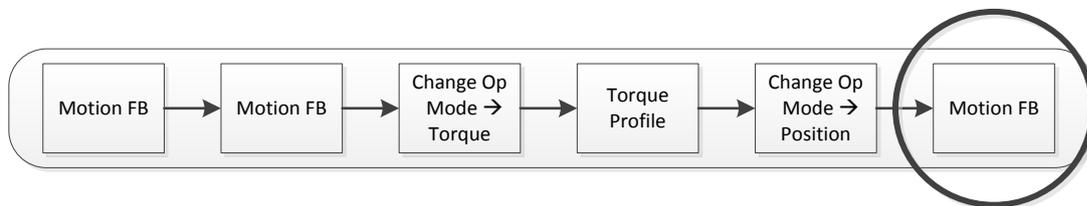
The operational mode is automatically changed to Cyclic Torque after the end of motion. The Torque profile starts immediately at the end of the Change Operational Mode. The mode is changed back to Cyclic Position as soon as the torque profile ends.

C API

The API for the MMC_ChangeOpModeEx is as follows:

int MMC_ChangeOpModeEx(

        IN MMC_CONNECT_HNDL hConn,

        IN MMC_AXIS_REF_HNDL hAxisRef,

        IN MMC_CHANGEOPMODE_EX_IN* pInParam,

        OUT MMC_CHANGEOPMODE_EX_OUT*pOutParam)

The Input structure:
```
typedef struct
{
        double dbInitModeValue;
        MC_EXECUTION_MODE eExecutionMode;
        unsigned char ucMotionMode;
unsigned char ucSpare[19];
}MMC_CHANGEOPMODE_EX_IN;
```

Where:

double dbInitModeValue – is the initial value of a specific parameter in the mode. Currently, states the initial torque command when the user chose to move to Cyclic Torque.

MC_EXECUTION_MODE eExecutionMode - Either eMMC_EXECUTION_MODE_IMMEDIATE or
        eMMC_EXECUTION_MODE_QUEUED . If the user chose the immediate option, this can be done when the axis is not in motion, and the value is sent as an SDO (same as previous function). If the queued option was chosen – the data is sent as process data over the Ethercat network. Please note: For this option, the user must have mapped the 0x6060 variable.

unsigned char ucMotionMode – The desired operational mode to move to.

C++ API

void CMMCSingleAxis::SetOpMode(OPM402 eMode, MC_EXECUTION_MODE eExecutionMode, double dbInitModeValue) throw (CMMCException)

This is an overloaded function. Calling the currently supported SetOpMode() function will always call the previous API.

Restrictions:

- Cannot be called when in Error State
- Not supported for Interpolated position motion mode when using Ethercat,
- All Cyclic motion modes (Position, Velocity, Torque) are supported in Ethercat only.
- Profile Velocity, Profile Position, Homing modes – Not supported for virtual axes.
- Profile Torque mode is not yet supported,
- Axis must not be part of an enabled Group.
- Operational mode cannot change when the axis is in pending state
- When moving to Torque mode – the desired torque cannot be larger than the maximum desired torque
- Cannot change operational mode when the axis is in the middle of a previous operational mode change,
- Queued mode is supported in Ethercat only.
- All variables were properly mapped using the Ethercat configuration tool.

1.14 Access to the GMAS Process Image – PI

New API's were introduced in this version in order to allow us to directly access the Process Image, and change the data. This does not apply of course to variables that were mapped and are used by the GMAS (Such as Target Position), but more to 3rd party IO vendors, or additional drive parameters that are used (Such as the Fast Reference Parameter in the drive).

When configuring the network, you are to notice 2 important columns within the process Image Folder:

Master	Quick Settings	Process Image	Cyclic	Distributed Clocks			
Name		Type	Bit Size	PI Offset	Value	Var Offset	Alias
Input variables (size 30bytes, length 30bytes)							
a01.Inputs.Position actual value		DINT	32	0	0	0	I0x6064.0
a01.Inputs.Digital Inputs		DINT	32	32	0	1	I0x60FD.0
a01.Inputs.Status word		UINT	16	64	0	2	I0x6041.0
a02.Inputs.Position actual value		DINT	32	80	0	0	I0x6064.0
a02.Inputs.Digital Inputs		DINT	32	112	0	1	I0x60FD.0
a02.Inputs.Status word		UINT	16	144	0	2	I0x6041.0
a03.Inputs.Position actual value		DINT	32	160	0	0	I0x6064.0
a03.Inputs.Digital Inputs		DINT	32	192	0	1	I0x60FD.0
a03.Inputs.Status word		UINT	16	224	0	2	I0x6041.0
Output variables (size 30bytes, length 30bytes)							
a01.Outputs.Target Position		DINT	32	0	0	0	O0x607A.0
a01.Outputs.Digital Outputs		DINT	32	32	0	1	O0x60FE.1
a01.Outputs.Control word		UINT	16	64	0	2	O0x6040.0
a02.Outputs.Target Position		DINT	32	80	0	0	O0x607A.0
a02.Outputs.Digital Outputs		DINT	32	112	0	1	O0x60FE.1
a02.Outputs.Control word		UINT	16	144	0	2	O0x6040.0
a03.Outputs.Target Position		DINT	32	160	0	0	O0x607A.0
a03.Outputs.Digital Outputs		DINT	32	192	0	1	O0x60FE.1
a03.Outputs.Control word		UINT	16	224	0	2	O0x6040.0

- Alias – The name of the variable being used.
- Var Offset – the Index of the variable being used

After adding a parameter via the FMMU page, the list of parameters will be updated accordingly. The user is to 'remember' the index / alias and the type of the variable – and can now access it from the user program.

There are mandatory parameters that must be mapped when working with a DS402 device:

- Control word 0x6040_0

- Status word 0x6041_0

- Target position 0x607A_0

- Position actual value 0x6064

'C' API

The following APIs were added in order to support this feature:

- MMC_GetPIVarInfo

Read

- MMC_ReadPIVarBOOL
- MMC_ReadPIVarChar
- MMC_ReadPIVarUChar
- MMC_ReadPIVarShort
- MMC_ReadPIVarUShort
- MMC_ReadPIVarInt
- MMC_ReadPIVarUInt
- MMC_ReadPIVarFloat
- MMC_ReadPIVarRaw
- MMC_ReadPIVarLongLong
- MMC_ReadPIVarULongLong
- MMC_ReadPIVarDouble
- MMC_ReadLargePIVarRaw

Write

- MMC_WritePIVarBOOL
- MMC_WritePIVarChar
- MMC_WritePIVarUChar
- MMC_WritePIVarShort
- MMC_ReadPIVarUShort
- MMC_WritePIVarInt
- MMC_WritePIVarUInt
- MMC_WritePIVarFloat
- MMC_WritePIVarRaw

- MMC_WritePIVarLongLong

- MMC_WritePIVarULongLong

- MMC_WritePIVarDouble

- MMC_WriteLargePIVarRaw

C++ API

```cpp
void EthercatWritePIVar(unsigned short usIndex, unsigned char ucByteLength, unsigned char
pRawData[PI_REG_VAR_SIZE]) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex,       unsigned short usByteLength, unsigned
char pRawData[PI_LARGE_VAR_SIZE]) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex, bool bData) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex,        signed char cData) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex, unsigned char ucData) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex, unsigned short usData) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex, short sData) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex, unsigned int uiData) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex, int iData) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex, float fData) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex, double dbData) throw (CMMCException);
void EthercatWritePIVar(unsigned short usIndex, unsigned long long ullData) throw
void EthercatReadMemoryRange(unsigned short usRegAddr,       unsigned char ucLength, unsigned
char pData[ETHERCAT_MEMORY_READ_MAX_SIZE]) throw (CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, unsigned char
ucByteLength, unsigned char pRawData[PI_REG_VAR_SIZE]) throw (CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, unsigned short
usByteLength, unsigned char pRawData[PI_LARGE_VAR_SIZE]) throw (CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, bool &bData) throw
(CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, signed char &cData)
throw (CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, unsigned char
&ucData) throw (CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, unsigned short
&usData) throw (CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, short &sData) throw
(CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, unsigned int
&uiData) throw (CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, int &iData) throw
(CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, float &fData) throw
(CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, double &dbData)
throw (CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, unsigned long long
&ullData) throw (CMMCException);
void EthercatReadPIVar(unsigned short usIndex, unsigned char ucDirection, long long &llData)
void EthercatPIVarInfo(unsigned short usPIVarIndex, unsigned char ucDirection, NC_PI_ENTRY
&VarInfo) throw (CMMCException);
```

Example

First you need to know the variable offset from the EtherCAT configuration after adding the object:

So in our example it is 3.

The input of the **`EthercatReadPIVar`** are : var offset , direction (is it input or output), variable that will hold the reading result

```
unsigned short iIndex = 3; //--→ the var offset

unsigned int iValue;       // -→will hold the value we are reading

aRef.EthercatReadPIVar(iIndex,ePI_INPUT,iValue); -→ ePI_INPUT meaning
```
this is an input variable.

1.15 Access to Drive Fast Reference 0x2005 variable

After configuring to work with the drive Fast Reference variable via the FMMU page within the GMAS configuration tool – you will see the following:
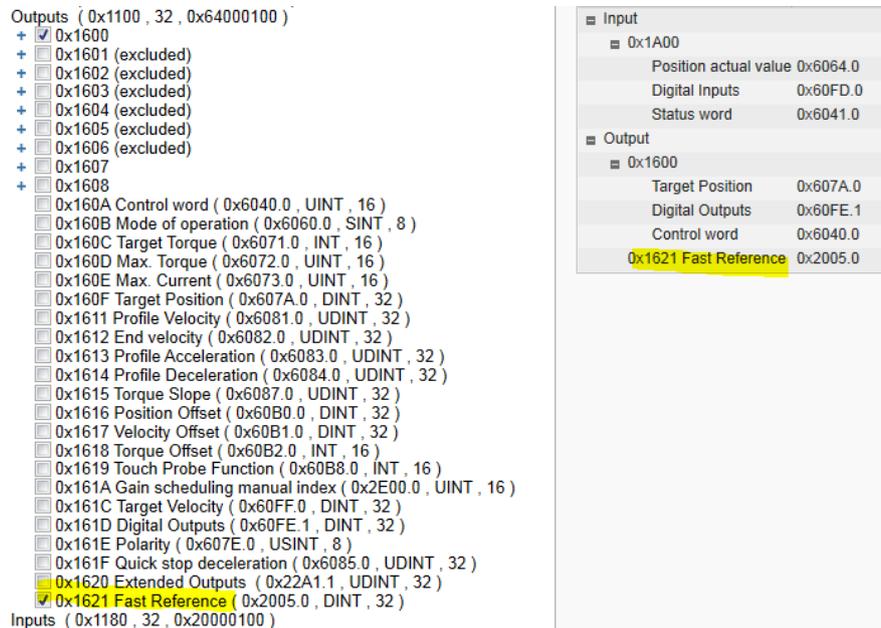
Using this variable allows the user to send an additional parameter to the drives 0x2005 parameter.

Sending this parameter can be done in 2 different modes:

- Mode0 – The value defined in the GMAS FAST_REFERENCE reflects what is automatically downloaded to the drives 0x2005 variable:
    - High 16 bit of the GMAS FAST_REFERENCE variable reflects the AxisReference to be used.
    - Lower 16 bits of the GMAS FAST_REFERENCE reflects the parameter to be sent:
        - 0 – TargetPosition
        - 1 – ActualPosition
        - 2 – S Position – along the path of a vector axis.

    This mode allows us to send a position of a specific axis to an additional physical axis in the system. For instance, a virtual axis can be used to generate a profile while its position is sent to the 0x2005 object which uses it as a pulse/dir axis.

- Mode1 - Whatever is written (manually) to the GMAS FAST_REFERENCE variable, is downloaded to the drive.

Interface

The standard GMAS parameters mechanism is to be used in order to access the following parameters:

- FAST_REFERENCE
- FAST_REFERENCE_MODE

1.16 New Home On Block API

The latest Drive versions introduced a new Home On Block (Methods -1, -2) homing methods. (Please refer to the drive manual).

In order to support the additional requirements, we added a new API called:

MMC_HomeDS402ExCmd

```
/////////////////////////////////////////////////////////////////////////////
/// \fn int MMC_HomeDS402ExCmd(
///                         IN MMC_CONNECT_HNDL hConn,
///                         IN MMC_AXIS_REF_HNDL hAxisRef,
///                         IN MMC_HOMEDS402EX_IN* pInParam,
///                         OUT MMC_HOME_OUT* pOutParam)
/// \brief This function  send Home command to MMC server for specific Axis.
/// \param  hConn - [IN] Connection handle
/// \param  hAxisRef - [IN] Axis Reference handle
/// \param  pInParam - [IN] Pointer to Home input parameters
/// \param  pOutParam - [OUT] Pointer to Home output parameters
/// \return  return - 0 if success
///                        error_id in case of error
/////////////////////////////////////////////////////////////////////////////
int MMC_HomeDS402ExCmd(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_AXIS_REF_HNDL hAxisRef,
            IN MMC_HOMEDS402EX_IN* pInParam,
            OUT MMC_HOME_OUT* pOutParam)
```

Where the new parameters (highlighted in yellow) were added to the Input Structure:

```
/////////////////////////////////////////////////////////////////////////////
/// \struct MMC_HOMEDS402EX_IN
/// \brief Home DS402EX Command input data structure.
/////////////////////////////////////////////////////////////////////////////
typedef struct mmc_homeds402ex_in
{
    double dbPosition;
    double dbDetectionVelocityLimit;
    float fAcceleration;
    float fVelocityHi;    //speed search for switch
    float fVelocityLo;    //speed search for zero
    float fDistanceLimit;
    float fTorqueLimit;
    MC_BUFFERED_MODE_ENUM eBufferMode;
    int uiHomingMethod;
    unsigned int uiTimeLimit;
    unsigned int uiDetectionTimeLimit;
    unsigned char ucExecute;
    unsigned char ucSpares[32];        //future usage
} MMC_HOMEDS402EX_IN;
```

When calling methods -1. -2 (Home on block) – Please use the new API. The previous API (HomeDs402Cmd) is no longer supported for these methods.

1.17 Kill Repetitive Support

A new API was introduced in this version release:

MMC_KillRepetitive

The API is responsible for stopping the ongoing repetitive motion at the end of the ongoing repetitive motion. As opposed to the *Stop* or *Halt* commands, the axis will not stop in between repetitive segments.

1.18 Modbus Client Library

The current G-MAS version release includes a Modbus client library, that enables the user to communicate from the user program, to Modbus server devices.

The user may do this Synchronously – Meaning, the library will wait for a reply and will not continue until a reply returned – or Asynchronously – the user code will continue to run before a reply was received from the server.

Defining an instance of the Modbus class

```
/* GMAS / Mbus Async. Services Class            */
MbusMasterAsyncServices mbusProtocol;
```

Opening a Modbus Client session

```
char *hostIp = "192.168.1.4";

result = mbusProtocol.LopenProtocol(hostIp);
if (result != FTALK_SUCCESS)
    {
        fprintf(stderr, "Error opening protocol: %s!\n",
        getBusProtocolErrorText(result));
        exit(EXIT_FAILURE);
    }
```

Writing ASYNC data (Multiple Registers)

```
#define sNUMREG 10      /* Short Number of Registers       */
short   WdataArr[sNUMREG];

WdataArr[1] = 0x1111;
WdataArr[2] = 0x2222;

/* Write to Address 100 from WdataArr sNUMREG registers */
/* No Block operation.                          */
result = mbusProtocol.LwriteMultipleRegisters(1, 100, WdataArr, sNUMREG);
if (result == FTALK_SUCCESS)
{
}
else
{
    ON_ERR_CODE()
}
```

Reading ASYNC data (Multiple Registers)

```
short   RdataArr[sNUMREG];
bool    ReadyForNewCommand;

/* Read from Address 100 to RdataArr sNUMREG registers  */
/* No Block operation.                          */
result = mbusProtocol.LreadMultipleRegisters(1, 100, RdataArr, sNUMREG);
```

```
if (result == FTALK_SUCCESS)
{
}
else
{
    ON_ERR_CODE()
}
```

Reading ASYNC data (Multiple Registers) – Waiting for Previous Read to finish

```
bool ReadyForNewCommand = false ;
do
{
/* No Block operation.                            */
    mbusProtocol.IsReady(&ReadyForNewCommand);
} while(ReadyForNewCommand == false);


/* Read from Address 100 to RdataArr sNUMREG registers  */
/* None Block operation.                           */
result = mbusProtocol.LreadMultipleRegisters(1, 100, RdataArr, sNUMREG);
if (result == FTALK_SUCCESS)
{
}
else
{
    ON_ERR_CODE()
}
```

Block until last ASYNC finished

```
/* Block till last operation Mbus servic done.     */
mbusProtocol.WaitForReady();
```

Write SYNC

```
result = mbusProtocol.LwriteMultipleRegisters(1, 100, WdataArr, sNUMREG,
MbusMasterSync);
if (result == FTALK_SUCCESS)
{
}
else
{
    ON_ERR_CODE()
}
```

Read SYNC

```
result = mbusProtocol.LreadMultipleRegisters(1, 100, RdataArr, sNUMREG,
MbusMasterSync);
if (result == FTALK_SUCCESS)
{
}
else
{
      ON_ERR_CODE()
}
```

1.19 User Parameters XML File Write support

The current version release supports the capability of creating and writing to an XML file.

The following CPP API's were added to the MMCUserParams.h file:

```
/* ======================= */
/* Update/add XML Tree data */
/* ======================= */
/* Functions Write XML file identifier *\
\* ================================= */
/* Create new XML tree in Ram. E.g when no file nore open...        */
/* with this function the user start build XML tree from scrach.*/
/* Should be the first write in sequnce of build new TREE.         */
/*                                                                 */
/* Write (null terminated string) to XML file root (xsi ID         */
/* values) pAtt1 & xsi Location (pAtt2),                           */
/* Eg:                                                             */
/* pAtt1 = "http://www.w3.org/2001/XMLSchema-instance"             */
/* pAtt2 = "proposed.xsd"                                          */
/* Write XML file lines:                                           */
/* <root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"     */
/* xsi:noNamespaceSchemaLocation="proposed.xsd">                   */

int PutXmlFileRoot  (char* pAtt1=UPXML_DEF_ROOT_ATT1, char* pAtt2=UPXML_DEF_ROOT_ATT2)
              throw (CMMCException);
/* Write (null terminated string) to XML "file description name"*/
/* (pAtt1) and XML file ver (pAtt2).                                      */
int PutXmlFileDescrp(char*    pAtt1,  char* pAtt2)            throw (CMMCException);

/* Write ONE parameter of type Double        */
int Write(char* pLvl1Valu,char* pLvl2Valu,char* pLvl3Name,double dWrVal)throw (CMMCException);

/* Write ONE parameter of type Long   */
int Write(char* pLvl1Valu,char* pLvl2Valu,char* pLvl3Name,long lWrVal) throw (CMMCException);

/* Write ONE parameter of type bool. */
int Write(char* pLvl1Valu,char* pLvl2Valu,char* pLvl3Name,  bool bWrVal) throw
(CMMCException);

/* Write ONE parameter of type string        */
int Write(char* Lvl1Valu,char* pLvl2Valu,char* pLvl3Name, char* pStr, long lLen) throw
(CMMCException);

//
/* Functions for Write Array of parameters values *\
                                                  \*
============================================= */
//
/* Write Array of parameter of type Double   */
int WriteArr(char* pLvl1Valu, char*  pLvl2Valu,char* pLvl3Name,
          double dWrVal[],   unsigned int   iActWrElm)   throw (CMMCException);

/* Write Array of parameter of type Long     */
int WriteArr(char* pLvl1Valu, char* pLvl2Valu, char* pLvl3Name,long lWrVal[],   unsigned int
iActWrElm)      throw (CMMCException);
```

After writing the desired parameters, the user must save the file by sending the *Save* command:

```
/* Be aware: 1. Default file name for read is different from default*/
/* name for save. 2. File is open for update but not save on exit  */
/* (or close)  but with explicit save command.                     */
int Save (char* cFileName=DEFAULT_XML_WFILE_NAME, char* cFilePath=DEFAULT_XML_WFILE_PATH)throw
(CMMCException);
```

1.20 Ability to Start / Stop User applications running on the GMAS

A new API was added that supports the ability to perform actions on the GMAS file system:

- Start a *.gexe application (User application).
- Stop a User application
- Returns whether an application is currently running or not.
- Returns whether an application exists on the file system, or not.
- Removes a file from the file system.

A few notes:

- Applications that are started are run in user context.
- If no path is stated in the path parameter, the default is the /mnt/jffs/usr location.

API

```
////////////////////////////////////////////////////////////////////////////
/// \fn int MMC_UserCommandControl(
///                             IN MMC_CONNECT_HNDL hConn,
///                             IN MMC_MOVETORQUE_IN* pInParam,
///                             OUT MMC_MOVETORQUE_OUT* pOutParam)
/// \brief This function send Move torque command to MMC server for specific Axis.
/// \param  hConn - [IN] Connection handle
/// \param  hAxisRef - [IN] Axis Reference handle
/// \param  pInParam - [IN] Pointer to Move Torque input parameters
/// \param  pOutParam - [OUT] Pointer to Move Torque output parameters
/// \return    return - 0 if success
///                     error_id in case of error
////////////////////////////////////////////////////////////////////////////

MMC_LIB_API int MMC_UserCommandControl(
        IN MMC_CONNECT_HNDL hConn,
        IN MMC_USRCOMMAND_IN* pInParam,
        OUT MMC_USRCOMMAND_OUT* pOutParam)
```

Where:

```
typedef struct
{
        MC_COMMAND_OPERATION eUsrCommandOp;
        char cUserCommand[256];
        unsigned char ucSpare[20];
} MMC_USRCOMMAND_IN;
```

The operations enabled are:

```
typedef enum command_operatioin
{
        eMMC_COMMAND_OPERATION_START =0,
        eMMC_COMMAND_OPERATION_STOP =1,
        eMMC_COMMAND_OPERATION_ISRUNNING =2,
        eMMC_COMMAND_OPERATION_ISEXIST =3,
        eMMC_COMMAND_OPERATION_REMOVE =4,
}MC_COMMAND_OPERATION;
```

1.21 FULL IEC Support

FULL IEC implementation was implemented in this GMAS version, including the following commands:

- MC_RESETASYNC – Equivalent to MC_ResetAsync API.
- MC_MOVETORQUE - Equivalent to MC_MoveTorque API.
- MC_HOMEDS402EX - Equivalent to MC_HomeDS402Ex API.
- MC_CHANGEOPMODEEX - Equivalent to MC_ChangeOPModeEx API.
- MC_READPIVARUINT - Equivalent to MC_ReadPIVarUint API.
- MC_READPIBYINDX - Equivalent to MC_ReadPIByIndex API.
- MC_WRITEPIBYINDX - Equivalent to MC_WritePIByIndex API.
- MC_WRITEPIRAWBYINDX - Equivalent to MC_WritePIRAWByIndex API.
- MC_GETPIINFO - Equivalent to MC_GetPIInfo API.
- MC_READPIRAWBYINDX - Equivalent to MC_ReadPIRawByIndex API.

The ReadPIByIndex and the WritePIByIndex are the only Read/Write PIs in the set of APIs as the functions receive an 'ANY' parameter enabling to receive any parameter type:
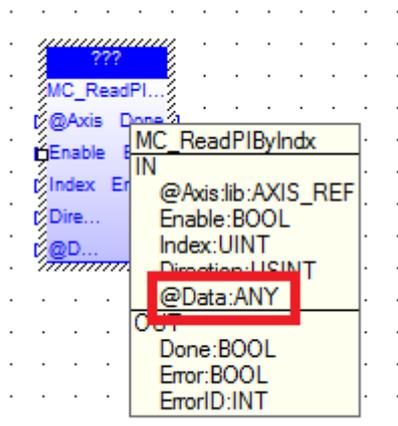
Release Notes - New Firmware Version for GMAS

Version 1.1.2.0 and Library Update 245

1. General

The "*uImage_v1.1.2.0.B4_2013_7_3.gms*" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

16. Administrative Function Block Support
    o WriteGroupParameters
    o Dwell
    o WaitUntilCondition Function Block.
17. Super Imposed motions.
18. VxWorks library interfaces.
19. Polynomial Function Block.
20. Additional Ethercat drive and IO support . Sanyo Denki, Omron, Beckhoff IO devices.
21. Additional Transition type for Blended ACS motion.
22. PVT at GMAS level. Unlimited number of points.
23. CAN configuration interfaces enabling PDO ,SDO configuration via resource file.
24. Ethernet/IP CPP libraries.
25. TCP/IP and UDP CPP libraries.
26. Enhanced Ethercat support. Enabling up to 8 axes at 500us
27. More events in GMAS for asynchronous operations. User is notified at end of operation. No need to poll
28. Enhanced personality file
29. Enhanced Error Correction Support
30. Additional event once an axis finished its initialization process.
31. Enhanced (we changed API) PDOInfo – returning more information regarding a mapped PDO.
32. GeneralPDOConfiguration – now supported for DS406 devices.
33. Ability to Load User Application Parameters from dedicated XML file.
34. IEC – Full implementation.
35. Enhanced Version Read.
36. New Kinematics in GMAS
    o Delta Robot Support.
    o Additional Kinematic Transformation Function definitions
37. Ethernet/IP support in IEC.
38. Async Reset Command.
39. Additional Parameters
    o SW
    o CW
    o Operation Mode
    o Digital Outputs
    o Digital Inputs
    o Speed Override
    o Last Node Init Error
    o Last SDO Abortion Error
    o Last System Error

40. Fast Reference Support to drive
41. Bug Fixes in this version.

1.1 New Administrative Function Blocks

New Administritive Function Blocks were introduced in this version. These function blocks operate in parallel to the ongoing motion blocks.

PLCopen introduced a new parameter in the WriteParameter Function block stating the FB execution mode:
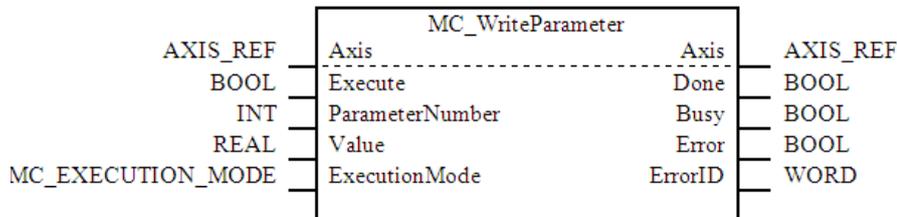
- **Immediately** – Will occur as soon as the Function Block is called.
- **Queued** – Will occur when the previous function block ended.
- Until now, the function block execution was always *Immediate.* This meant – it was difficult to synchronize the function call to the ongoing motion.
- Now – User can precisely change parameters (Also parameters that are mapped to communication), IO's, Maximum Torque, at the end of a specific function block.

The following new Administrative Function Blocks were added in this version:
- *WriteGroupOfParametersFB – Ability to write up to 5 parameters in 1 function block call,* between 2 motion blocks*. This allows the user to accurately modify GMAS parameters between motion segments. (Not only parameters, but also IO's, Speeds, etc ...). These parameters do not necessarily have to belong to the axis in motion. We can precisely modify an IO of axis#2, as soon as axis #1 blends its motion to an ongoing Function Block.*
- *DwellFB – the ability to a precise wait between ongoing function blocks was added in this version.*

In the following example, the 2nd MoveAbsolute will not be called until the 2 dwells and SetParameter finished:
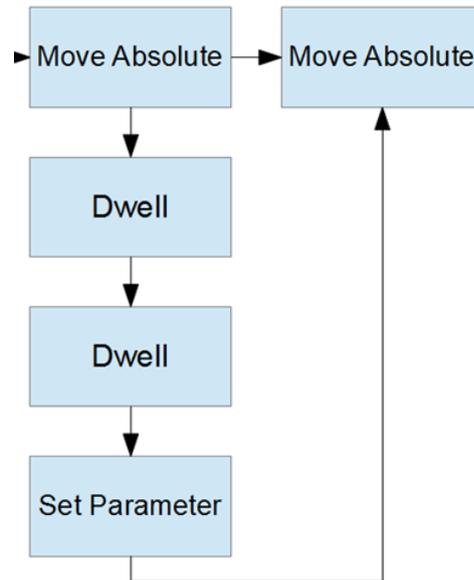
2D Motion with Precise Torque Change

A defined sequence where we want to set high current to an axis performing a 2D motion at precise location.

① - 2D linear move.
② - Set high max current value.
③ - 2D linear move.
④ - Set low max current value.
⑤ - 2D linear move.
⑥ - 2D linear move.
The motion sequence performed with blending transitions between function blocks, all the sequence is performed without stopping.
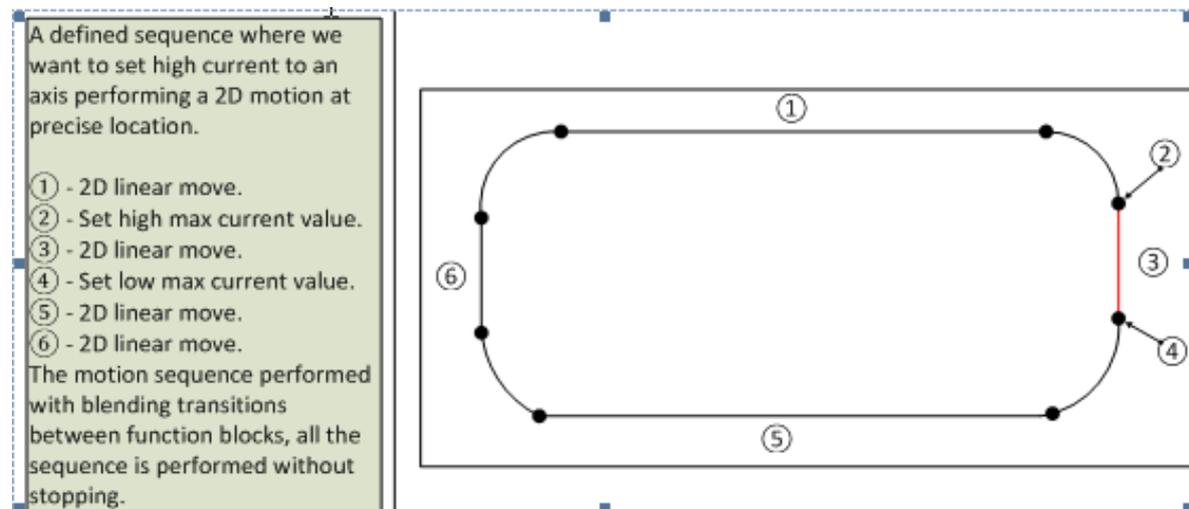
IO Change at precise location

Motion sequence where we want to turn light on, using digital output at a specific location.

① - 2D Circle motion.
② - Set digital output.
③ - 2D Circle motion.
④ - 2D linear move.
⑤ - Clear digital output.

The motion sequence is performed with blending transitions between function blocks, all the sequence is performed without stopping.
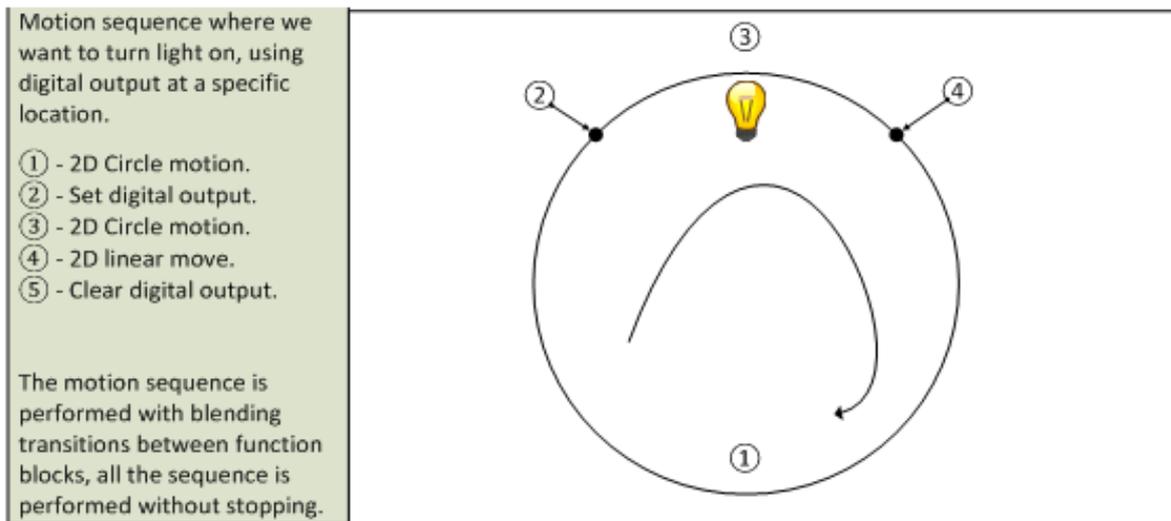
- *WaitUntilConditionFB-* Similar to the Dwell Function Block, but waits until a condition is fulfilled. The conditions are based on the list of the parameters supported by the GMAS parameters list. The user can perform any of the following conditions on the parameters, on any parameter type:

    - **EQUAL,**
    - **LOWER**
    - **HIGHER**
    - **LOWER_EQUAL**
    - **HIGHER_EQUAL**
    - **MASK_AND**

    The Condition does not necessarily need to belong to the axis queue it is operated on.

    This gives us the ability to synchronise numerous axes that are not part of a group, to begin motion together. It also gives us the ability to synchronise numerous G-MAS's on the network by starting motion once a specific bit on a shared IO is raised.

For instance:

If we need to synchronize the Motion Start between 30 axes, we can insert a condition FB to each axis and then insert some requested motion in the next Function Block.

Once the condition is fulfilled, all motions will start together.
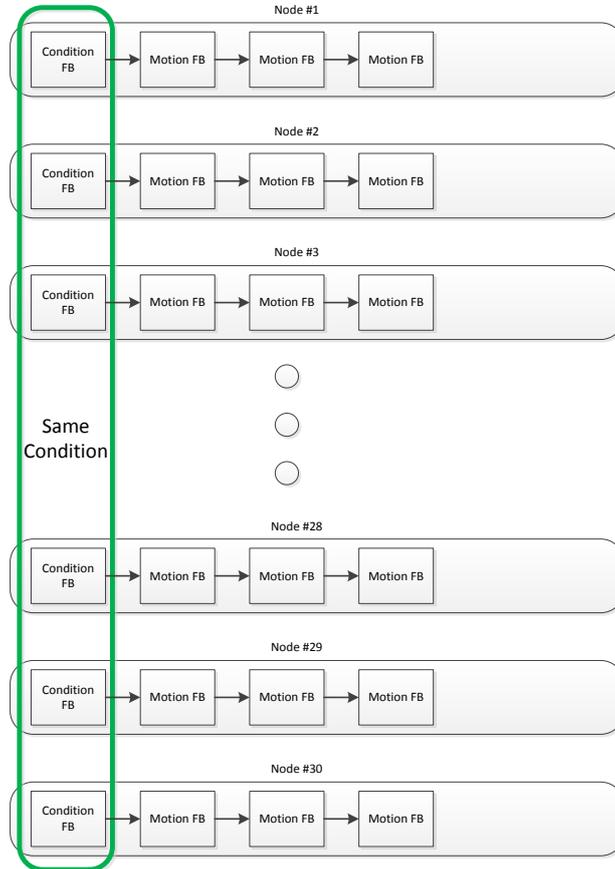
We can synchronize the axes / groups whilst the synchronization point defined using any type of parameter and any type of logical comparison. The compared parameters can be referred to any desired node (Single/Multi) you select:
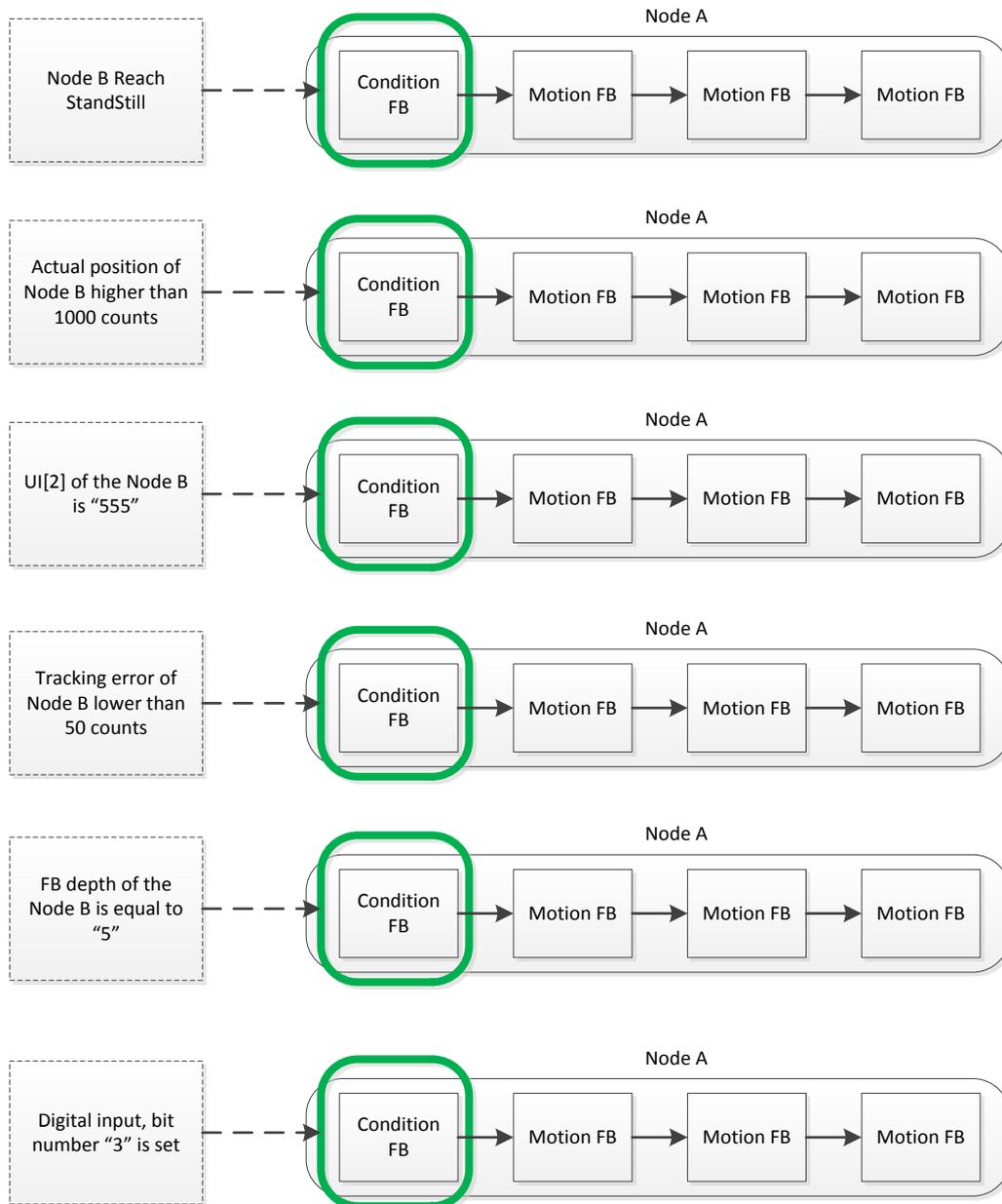
API's:

```
MMC_LIB_API int MMC_WriteGroupOfParameters(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_AXIS_REF_HNDL hAxisRef,
            IN MMC_WRITEGROUPOFPARAMETERS_IN* pInParam,
            OUT MMC_WRITEGROUPOFPARAMETERS_OUT* pOutParam)
```

- This API, writes a group of up to 5 parameters to the GMAS memory.
- Unlike the WriteBoolParameter, WriteParameter, WriteGlobal, etc … - this function receives double precision floating point values, irrelevant to the parameter type, and writes it to the appropriate memory space. The user does not need to know the memory type. This function is to be used in all new applications.
- The ucMode parameter defines whether the function is to be inserted as a buffered function block (will take affect when the previous function block ends) or immediately (may affect the ongoing motion as the Function Block is inserted immediately).
- Please refer to the user manual for extra information.

```
MMC_LIB_API int MMC_ReadGroupOfParameters(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_READGROUPOFPARAMETERS_IN* pInParam,
            OUT MMC_READGROUPOFPARAMETERS_OUT* pOutParam)
```

- This API does not actually belong to the administrative FB's, as it returns parameter values to the user, but was inserted here together with its sibling: **WriteGroupOfParameters**.

```
MMC_LIB_API int MMC_DwellCmd(IN MMC_CONNECT_HNDL hConn,
                IN MMC_AXIS_REF_HNDL hAxisRef,
                IN MMC_DWELL_IN* pInParam,
                OUT MMC_DWELL_OUT* pOutParam)
```

- This API performs a precise wait, according to the time defined.
- Can be inserted only to a motion that ends at velocity 0.
- Please refer to the user manual for additional information

```
MMC_LIB_API int MMC_WaitUntilConditionFB(IN MMC_CONNECT_HNDL hConn,
                                         IN MMC_AXIS_REF_HNDL hAxisRef,
                                         IN MMC_WAITUNTILCONDITIONFB_IN* pInParam,
                                         OUT MMC_WAITUNTILCONDITIONFB_OUT* pOutParam)
```

- ■ This API precise wait until a condition is fulfilled. For instance: Wait until position > 10000 on axis 'A' before starting a new motion on Axis 'B'.
- ■ Can be inserted only to a motion that ends at velocity 0.
- ■ Please refer to the user manual for additional information

Of course, all of the new administrative Function Blocks are available from within the EAS script manager:

a02.Wait Until Condition FB		
a02.Dwell		
a02.Write Group Parameters		

Misc	
Axis Name	a02
Execute	1
Wait Condition Parameters	
Reference Axis Name	a01
Parameter	MMC_ACT_POSITION_PARAM
ParameterIndex	0
Operation Type	MC_CONDITIONFB_OP_HIGHER
Value	512

1.2   Super Imposed Motions

Super Imposing a motion is the ability to impose an additional profiler to the ongoing motion. **It is very useful when the target position is changed on the fly.**

- This is widely used in cases where the end position is unknown and needs changes during the motion, without crossing the initial location. **We can change the target position without changing the current FB.** For instance – when a camera provides new target coordinates whilst the axis is already in motion.

- The imposed Position, Velocity, ACC/DC is added to the ongoing motion
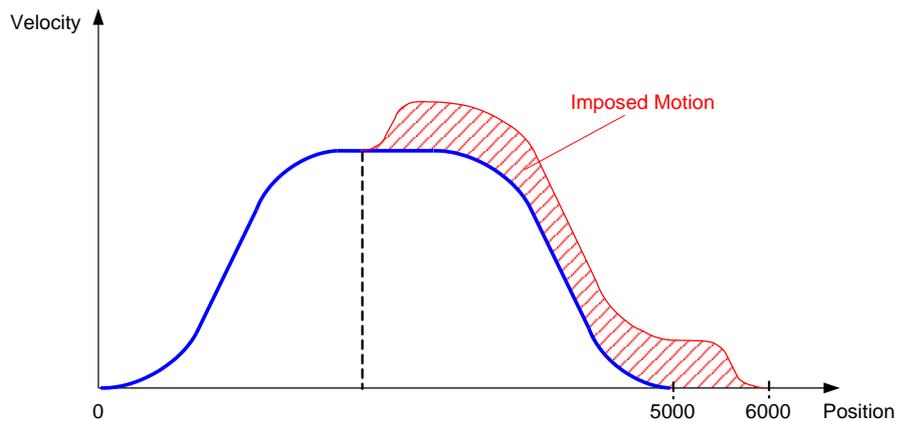
The Super Imposed functionality is provided by linking a master axis to a slave axis.

- Linked Master – This is the axis that 'receives' the position.

- Linked Slave – This is the axis that 'supplies' the position to the linked master.

API's:

```
MMC_LIB_API int MMC_AxisLink(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_AXIS_REF_HNDL hAxisRef,
            IN MMC_AXISLINK_IN* pInParam,
            OUT MMC_AXISLINK_OUT* pOutParam)
```

AxisLink

- Axis Link – link between master and slave.

    - Each axis can be linked only once.

    - Both axes must be in StandStill state.

    - Slave axis must be a virtual axis (currently).

    - The offset position in the slave should be "0".

```
MMC_LIB_API int MMC_AxisUnLink(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_AXIS_REF_HNDL hAxisRef,
            IN MMC_AXISUNLINK_IN* pInParam,
            OUT MMC_AXISUNLINK_OUT* pOutParam)
```

Axis UnLink – unlink the master-slave relation.

- Can be activated from any axis, master, slave.

- Automatically performed in power on procedure.

- Slave axis should be in non-motion state.

- The offset from the slave should be "0".

The Ability to perform SuperImposed motions is also part of the script manager.
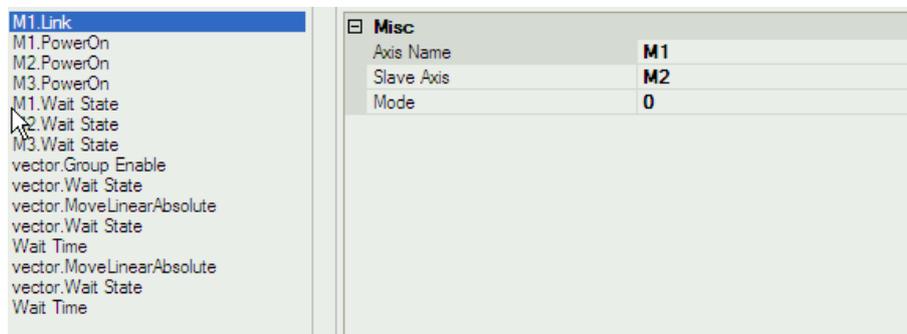
There is an ability to record Before / After Super Imposed is set:
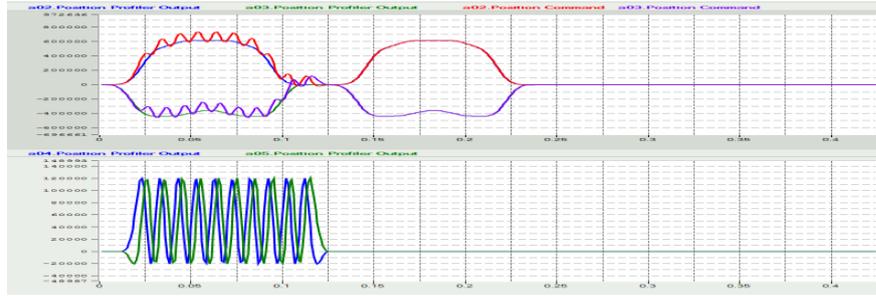
The StatusRegister variable also reflects whether an axis is in a 'Linked' – superimposed state, or not.

- Outputs – Status register

    - bit 16 – The axis is linked as master.

    - bit 17 - The axis is linked as slave.

Topology
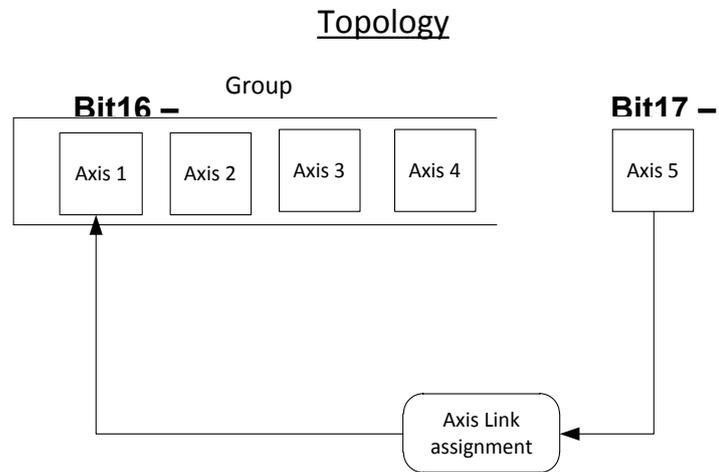
1.3　VxWorks library interfaces

In addition to the following host interfaces to the GMAS:

- .NET API
- 'C' and CPP library interfaces.

We released an additional ability to communicate via VxWorks Host operating systems.

1.4   Polynomial Function Block

Polynomial Function Blocks were introduced in order to improve the ability to build motion sequences with smooth velocity and AC / DC while performing transitions between motion blocks

- Linear
- Circular

The motion shape that is received is of 5th order polynomial order.

The Polynomial crosses 3 given points:

- Start position
- Auxiliary Position
- End Position
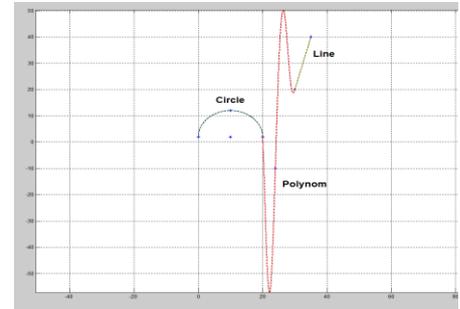
Notes Regarding the Polynomial Function Block:

- The auxiliary point can be any point in the 2D plane or 3D shape or in n-dimensional coordinate.
- The polynomial function block can change the vector velocity during the execution, as defined in the blended mode:
    - If the polynomial FB is first FB – start velocity and AC\DC is "0".

V = 0
AC\DC = 0

Polynom FB
Buffered

V = 0
AC\DC = 0

Motion FB
Buffered

V = 0
AC\DC = 0

Polynom FB
Buffered

V = Vblend
AC\DC = AC\DC Blend

Motion FB
Blended
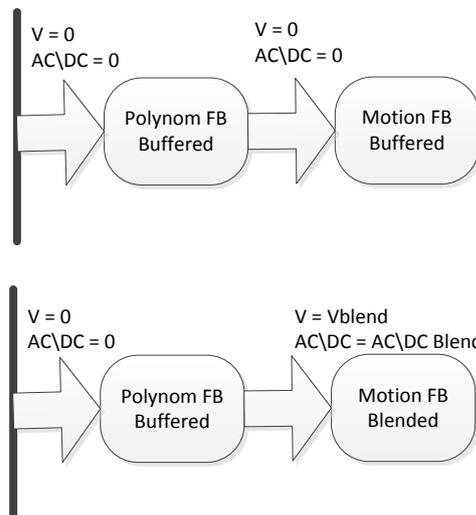
    - If the polynomial FB is last FB – end velocity and AC\DC is "0".

V = 0
AC\DC = 0

Motion FB → Polynom FB Buffered

V = Vblend
AC\DC = AC\DC Blend

Motion FB → Polynom FB Blended
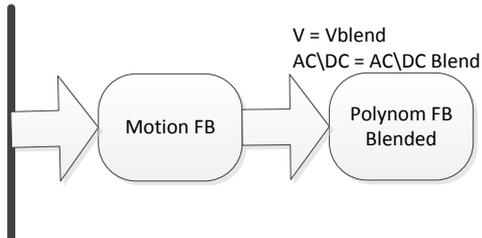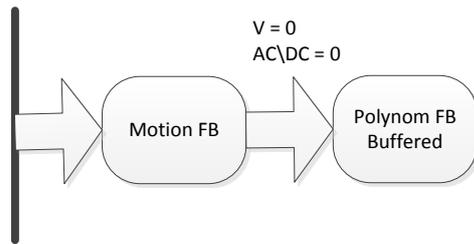
- o If the polynomial FB is in the middle of FB sequence –
  - Start kinematics defined by blended mode of the polynomial FB.
  - End kinematics defined by blended mode of the next FB.

V = 0
AC\DC = 0

V = 0
AC\DC = 0

Motion FB → Polynom FB Buffered → Motion FB Buffered

V = 0
AC\DC = 0

V = Vblend
AC\DC = AC\DC Blend

Motion FB → Polynom FB Buffered → Motion FB Blended

V = Vblend   V = 0
AC\DC = AC\DC Blend   AC\DC = 0

Motion FB → Polynom FB Blended → Motion FB Buffered

V = Vblend   V = Vblend
AC\DC = AC\DC Blend   AC\DC = AC\DC Blend

Motion FB → Polynom FB Blended → Motion FB Blended
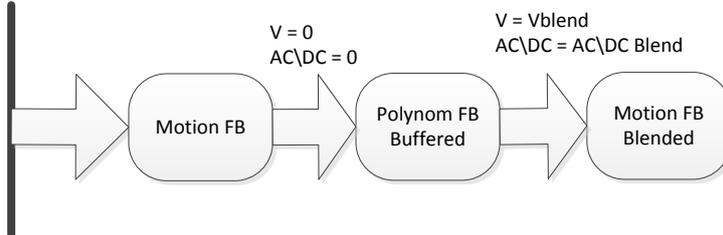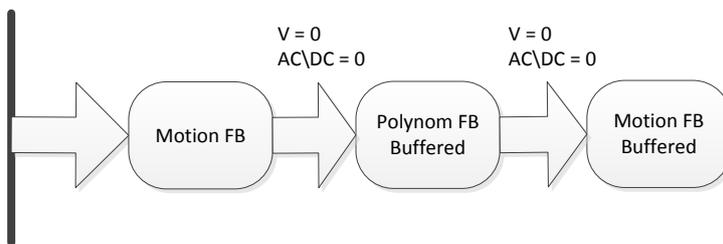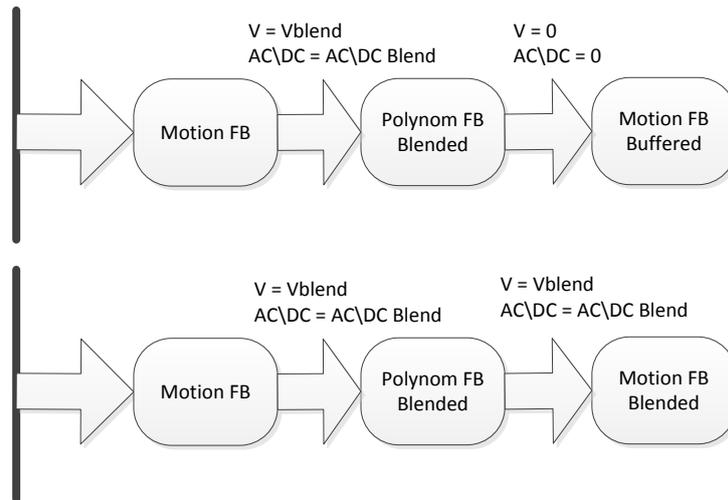
We support the case when only one Polynom FB is entered on queue.
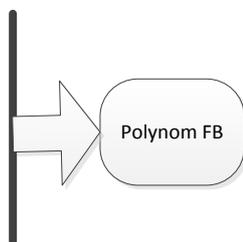
Polynom FB

- Currently, we do **not** support two consecutive polynomial FB's. In this case, the user will get an error.

Motion FB → Polynom FB → Polynom FB → Motion FB
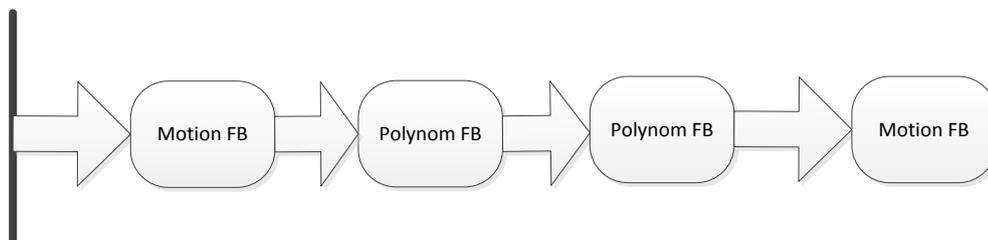
- Polynomial FB does not support any transition modes.
- Polynomial FB supports all possible buffer modes.
    - o Abortion.
    - o Buffered.
    - o Blended (4 modes).
- Polynomial FB is be supported in both coordinate systems, ACS and MCS.
- Polynomial FB can be inserted with execute bit "1" or "0".
- If we enter the polynom FB with execute bit "1", the next motion fb can be inserted only with buffered mode (no blendings).

- When we insert polynom FB the following limitations are checked:
  - SW limits – for aux and end points.
  - Velocity.
  - AC.
  - DC.
  - Jerk.
- Currently, As in all other motion FB's, we cannot add Polynom FB to table related FB's (we don't know the point of the table before it is done):
  - Spline.
  - PVT.
  - ECAM.
- Allowed PLC state:
  - Group not in Error.
  - Group not in stopping when current FB inserted in Abortion mode.
  - Group not in disabled.
  - Group is Valid.
  - All axes in group are valid:
    - Not disabled.
    - Not error.
- Currently, We do not provide the ability to insert Table related FB (Spline, PVT,ECAM) when the last inserted FB is polynom with execute bit "0".
- When the polynom fb is the last fb in queue we act as in other motion fb's:
  - Manage the Target Reached bit.
    - Target Radius.
    - Target Time.
  - When target is reached and the user enables the MotionEndedEvent, event will be sent to the user.
- We do not make validations of point closeness:
  - Start ←→ Aux.
  - Start ←→End.
  - End ←→ Aux.
  - Start ←→ Aux ←→ End.

Using a the Polynomial function block promises smoothness throughout the path as far as Position, Velocity, AC/Dc are concerned.

 For instance:

Start :   0,0
Line: 10k,10k
Polynom: Aux(14k,19k) End(20k,18k)
Circle (angle): Aux(30k,18k) Angle (-180)
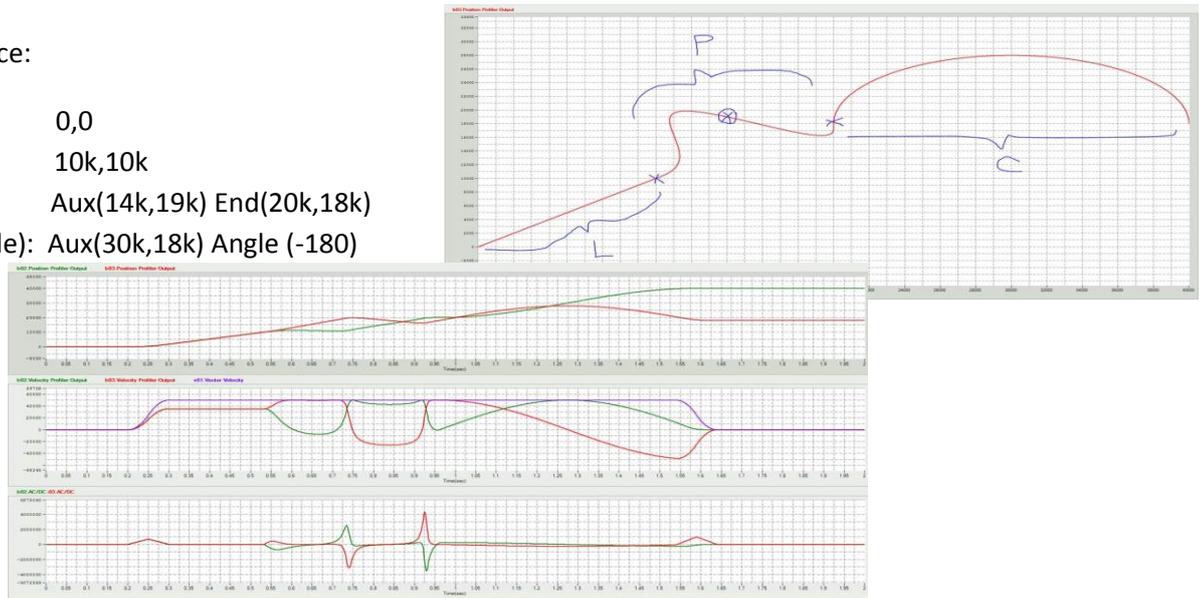
API's – C library:

```
MMC_LIB_API int MMC_MovePolynomAbsoluteCmd(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_AXIS_REF_HNDL hAxisRef,
            IN MMC_MOVEPOLYNOMABSOLUTE_IN* pInParam,
            OUT MMC_MOVEPOLYNOMABSOLUTE_OUT* pOutParam);


typedef struct
{
      double dAuxPoint[NC_MAX_NUM_AXES_IN_NODE];
      double dEndPoint[NC_MAX_NUM_AXES_IN_NODE];
      double dVelocity;
      double dAcceleration;
      double dDeceleration;
      double dJerk;
      MC_COORD_SYSTEM_ENUM eCoordSystem;
      MC_BUFFERED_MODE_ENUM eBufferMode;
      unsigned char ucSuperimposed;
      unsigned char ucExecute;
}MMC_MOVEPOLYNOMABSOLUTE_IN;


typedef struct
{
      unsigned int    uiHndl;        ///< Function Block Handle.
      unsigned short usStatus;       ///< Returned command status.
      short usErrorID;               ///< Returned command error ID.
}MMC_MOVEPOLYNOMABSOLUTE_OUT;
```

API - CPP Library

As in other group's motion related functions, we created several types of overloaded functions.

All move polynomial functions are implemented in the "CMMCGroupAxis" Class.

```
int MovePolynomAbsolute(MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE)


int MovePolynomAbsolute (float fVelocity,
                   MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE)

int MovePolynomAbsolute (float fVelocity,
                   double dbAuxPosition[NC_MAX_NUM_AXES_IN_NODE],
                   double dbEndPosition[NC_MAX_NUM_AXES_IN_NODE],
                   MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE)

int MovePolynomAbsolute (float fVelocity,
                   double dbAuxPosition [NC_MAX_NUM_AXES_IN_NODE],
                   double dbEndPosition [NC_MAX_NUM_AXES_IN_NODE],
```

```
            float fAcceleration,
            float fDeceleration,
            MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE)


int MovePolynomAbsolute (float fVelocity,
            double dbAuxPosition [NC_MAX_NUM_AXES_IN_NODE],
            double dbEndPosition [NC_MAX_NUM_AXES_IN_NODE],
            float fAcceleration,
            float fDeceleration,
            float fJerk,
            MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE)
```

1.5 Additional Ethercat drive and IO support

Extensive work was performed in order to support additional Slave devices with the GMAS Ethercat master. The following slaves are supported:

- Beckhoff Digital IO:
  - Input: EL1004
  - Output: EL2004
- Beckhoff Analogue IO:
  - Output: EL4031
  - Input: EL3104
- Sanyo Denki Drives
- Omron Drives.

These devices are of course supported by the Elmo Application Studio (EAS).
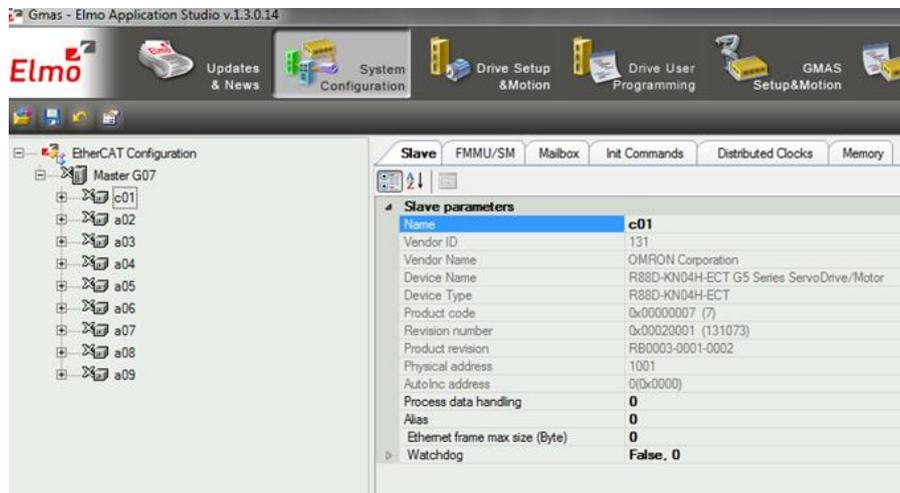
1.6 Additional Transition type for Blended ACS motions

Until now, in order to perform blended motions, they had to be part of an X-Y-Z Cartesian system.

In order to blend the motion, the user had to work in MCS, and select the Kinematic direction for each axis. Systems with no Kinematic direction, (as we call – N axes) – we could not perform blended motions, and all motions had to be buffered.

For instance, the following robot, does not represent a Cartesian System, but we would like to perform blended motions in ACS.

A new transition type was introduced: MC_TM_CORNER_DIST_CV_POLYNOM5_NAXES = 10

This mode is a Corner Distance, fifth order polynomial transition mode for ACS

All the user needs to do is define the above transition as part of an ACS motion, and the transition will be performed:

Here we can see 12 axes in blended motion, before the implementation (had to be buffered – Velocity reached zero between segments), and after:
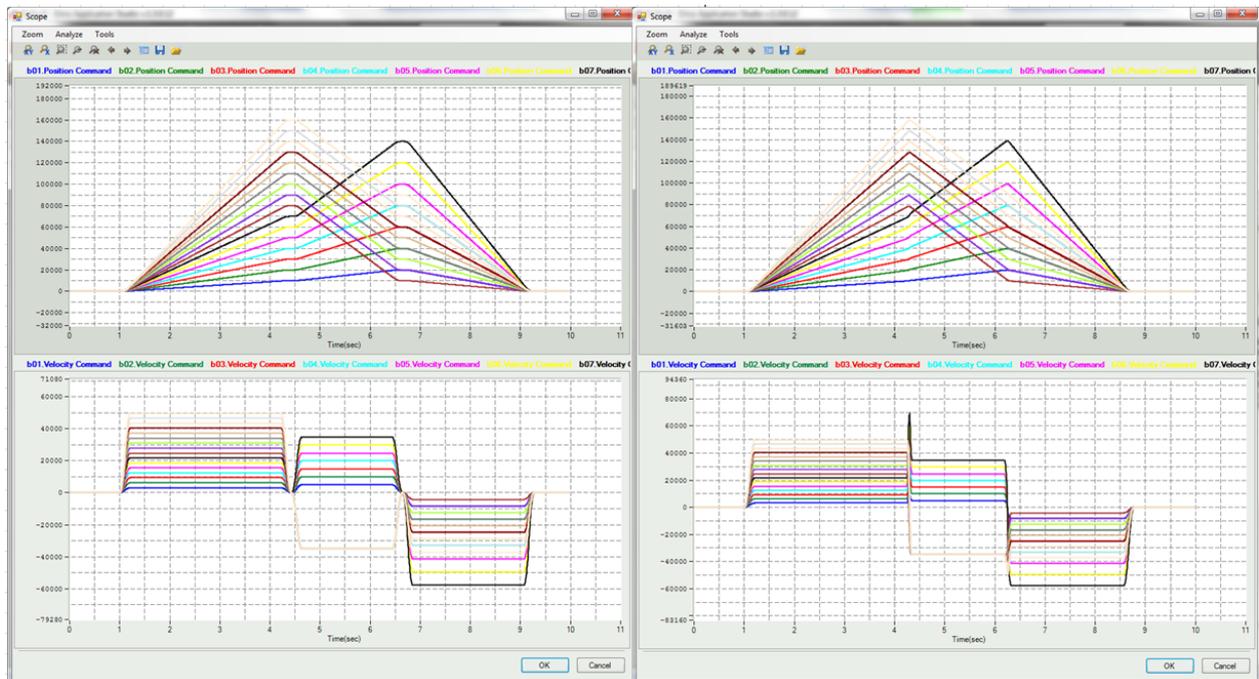
The following should be taken into account:

- When all axes move to same direction – the transition between the FB is in blended mode.
- When all axes move to the opposite direction – the FB will is inserted in Buffered mode.
- High value of transition parameter – the parameter will be saturated by 40% of the shortest segment

1.7   PVT at G-MAS level

Feature highlights:

Users can specify a prepared or on the fly path, up to 16 axes - with discrete position, velocity and time. The GMAS will interpolate (5th order) to create a smooth and contiguous path

- Single and Group axes supported.

- Absolute / Relative positions and timing.

- Large cyclic (option) PVT memory

  o Single axis > 40K points.

  o 3 Axes > 17K points.

  o 16 Axes > 3K points.

- Modes

  o Static mode – Based on a file or table points.

  o Dynamic mode (On the fly mode)– Based on table points, sent on the fly to GMAS. Buffer underflow event mechanism is available in this mode.
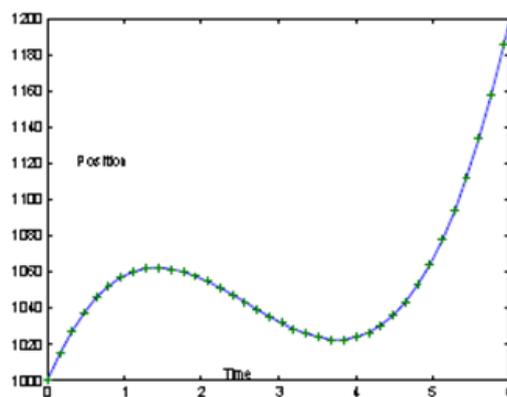
PVT describes the path given by position-velocity pairs per axis, and an optional time interval given per system (expressing as time per axis is unnecessary), and is applicable both for single and multi-axis motion.
The input data for PVT motion is stored as a table in the G-MAS RAM using a function. The PVT motion can operate in "normal" or cyclic modes. In cyclic mode, the profiler reaches the end of table and returns to the beginning.

Generally, PVT motion is defined by a set of points, and if a number of points are provided in advance of the present position, the profiler can build a 3rd degree polynomial by which to calculate the next position to be downloaded to the drive (Dynamic Mode). The path is calculated on-the-fly, and therefore all polynomial

coefficient calculations occur in the real-time module. The PVT motion does not require full data to execute, and only requires a minimal number of points.

Similar to splines, the input table is stored in the shared memory, but unlike splines, only the coordinates are stored, whereas the polynomial coefficients are calculated in the real-time module. The tables can be loaded via file, or via an N x M user-provided array. Optionally, a row or a few rows can be appended to the tables, but within reasonable constraints; for example, data cannot be appended to the currently segment, the profiler is operating on. The PVT function blocks are only applicable in NC cyclic/interpolated modes.

Each row of input data stored as a table in shared memory, represents the time, position, velocity and vector position. For example, if we have an m-axes setup, the nth row in the table will be generated according to the following format:

$$T_n \qquad P_{n1} \quad V_{n1} \quad P_{n2} \quad V_{n2} \quad \dots \quad P_{nm} \quad V_{nm}$$

As shown, T is similar for all axes, and therefore can be used only once.

The user can provide the data in several ways:

- Provide a file with a table containing the data points

- Provide an array containing data points, which can also be appended to an existing path within a given index

When the user decides to append points to an existing path, he has to be aware that:

- The table size is limited, and its maximum size should be known

- For dynamic Append only, when the number of points exceeds the table upper limit, the points will be appended to the beginning of the table.

- It is not possible to append to the current segment (presently, if the current index is X, the user can only append from X + 3).

During data loading, the points are read from the file. However, unlike splines, they are inserted into shared memory without pre-calculations.

Appending data to a table loaded from file is forbidden.

Every cycle the next segment is calculated forward. The function block includes a pointer to the current selected path data start, and includes all path-constant data, i.e. dimension, number of points, etc. In addition, the function block contains the current working index, so that if the user wishes to append points, the system will avoid appending the points at the current working index. When the PVT function block is in motion, no other function block can be inserted in any buffer mode, only via the STOP command (similar to spline behavior).

On The Fly Mode

The G-MAS also supports loading data to the G-MAS on-the-fly, i.e. the user can start the motion based on existing data, and the remaining data can be appended later. This concept allows the user to change the path

on-the-fly, taking the software constraints into consideration. At this moment, there are two appending sub-modes implemented; automatic and manual.

In automatic mode, the G-MAS will remember the last index where data was appended, and will append to this index next time.

In the manual mode, the user will have to provide the index, where he wishes to append points.

The number of points calculated is used in the real-time module to check that the end of path has not been reached, i.e. if 2000 points is allocated, and the user inserts just 1000 points, the 1000 points limit should not be traversed. In addition, in dynamic mode only, an underflow event can be sent when the index is close to the end of path.

Initializing The Data Table

Before appending points to a table, the user should initialize it. This can be achievd in two ways; by loading a table from file, or invoke a dedicated function, which will initialize the "constant" path parameters, like dimensions, maximal number of points, etc.
In addition, the user should select whether the appending is to be done statically or dynamically. If dynamic appending is selected, the user should choose an underflow threshold.

Loading Data

There are two modes of load data from array – Static and Dynamic. Static mode is loading the data from file mode, the table is loaded and no appending is allowed. Dynamic mode is cyclic by default, and real-time events will be generated upon crossing the underflow threshold. Each time the number of inserted points drops below the predefined threshold – an event is generated. Using the existing events mechanism – the user handles the overflow as necessary – the user can add points once an event occurred.

The user has control of two parameters during appending data to table; Whether appending in automatic mode? If no, then the second parameter is the index to append from. Based on these two parameters, the algorithm inserts the data safely.

If the axis/vector is in motion, an index delta is applied to maintain the calculated path (index delta = 3). If the difference between the current index and the start index to append is less than the delta, insertion is forbidden.
In cyclic mode the data is appended to the given index (automatically or manually), and when the data reaches the end of PVT segment, it is automatically appended to the beginning. In non-cyclic mode, where the data reaches the end of PVT segment, an error is returned.

Cyclic Mode

In order to support cyclic mode, a cyclic buffer is managed. The main constraint is that the size of the buffer appended must not exceed the table size, as then the end will run over the beginning. In addition, if the appending index is after the current index, the user should maintain a minimal delta:

Abs(current – start) < 3

If the index of appending is prior to the current index, the following should be maintained:

start + block size < current

API's

Please refer to User Manual for further information.

C – API's

```
MMC_LIB_API int MMC_InitTableCmd(IN MMC_CONNECT_HNDL hConn,
                       IN MMC_INITTABLE_IN* pInParam,
                       IN MMC_INITTABLE_OUT* pOutParam)


MMC_LIB_API int MMC_LoadTableFromFileCmd(IN MMC_CONNECT_HNDL hConn,
                       IN MMC_LOADTABLEFROMFILE_IN* pInParam,
                       OUT MMC_LOADTABLE_OUT* pOutParam)


MMC_LIB_API int MMC_AppendPointsToTableCmd(MMC_CONNECT_HNDL hConn,

                       MMC_APPENDPOINTSTOTABLE_IN* pInParam,

                       MMC_APPENDPOINTSTOTABLE_OUT* pOutParam)


MMC_LIB_API int MMC_GetTableIndexCmd(IN MMC_CONNECT_HNDL hConn,
                       IN MMC_GETTABLEINDEX_IN* pInParam,
                       OUT MMC_GETTABLEINDEX_OUT* pOutParam)


MMC_LIB_API int MMC_MoveTableCmd(IN MMC_CONNECT_HNDL hConn,
                       IN MMC_AXIS_REF_HNDL hAxisRef,
                       IN MMC_MOVETABLE_IN* pInParam,
                       OUT MMC_MOVETABLE_OUT* pOutParam)


MMC_LIB_API int MMC_UnloadTableCmd(MMC_CONNECT_HNDL hConn,
                       MMC_UNLOADTABLE_IN* pInParam,
                       MMC_UNLOADTABLE_OUT* pOutParam)
```

CPP API's

```
MC_PATH_REF CMMCMotionAxis::InitPVTTable(unsigned long ulMaxPoints,
     unsigned long ulUnderflowThreshold,
     unsigned char ucIsCyclic,
     unsigned char ucIsPosAbsolute,
     unsigned short usDimension,MC_COORD_SYSTEM_ENUM eCoordSystem) throw
(CMMCException)


MC_PATH_REF LoadPVTTableFromFile(char* pFileName,MC_COORD_SYSTEM_ENUM
eCoordSystem) throw (CMMCException);

virtual void MovePVT(MC_PATH_REF hMemHandle, MC_COORD_SYSTEM_ENUM
eCoordSystem) throw (CMMCException);

void UnloadPVTTable(MC_PATH_REF hMemHandle) throw (CMMCException);
```

```
void AppendPVTPoints(MC_PATH_REF hMemHandle,
     double dTable[NC_PVT_ECAM_MAX_ARRAY_SIZE],
     unsigned long ulNumberOfPoints,
     unsigned char ucIsTimeAbsolute = 0) throw (CMMCException);
void AppendPVTPoints(MC_PATH_REF hMemHandle,double

     dTable[NC_PVT_ECAM_MAX_ARRAY_SIZE],
     unsigned long ulNumberOfPoints,
     unsigned long ulStartIndex,
     unsigned char ucIsTimeAbsolute = 0) throw (CMMCException);


unsigned int GetPVTTableIndex(MC_PATH_REF hMemHandle)  throw
(CMMCException);
```

1.8 CAN configuration interfaces enabling PDO and SDO configuration via resource file

In order to configure CAN PDO's, the user needs to know the following APIs:

- MMC_CfgRegParamEvPDO3Cmd
- MMC_CfgRegParamEvPDO4Cmd
- MMC_SetSyncTimeCmd
- MMC_CfgEventModePDO3Cmd
- MMC_CfgEventModePDO4Cmd

In order to EAS the usage of these functions, the GMAS now supports these functions as part of the resource file. (Not yet supported in EAS).

In addition to PDO settings, the GMAS resource file also supports the ability to send SDO's to devices (DS301, DS402, DS406).

These SDO's can be sent in Preoperational or Operational modes

XML Format

The XML resource file has two sections located in the node section:

- PDO – one for each PDO mapping. Up to two TPDOs and two RPDOs.
- SDOs- each node may have a maximum number of 8 SDO sections (some in operational and some in pre operational mode)

A NAME element for SYNC was added in the global parameters section

The fields in the PDO paragraph and their values are:

PDO_NUM -two possible values "3" or "4"

DIRECTION- two possible values: "0" for RPDO or "1" for TPDO

CommPARAM- possible values: "1" for SYNC, "255" for ASYNC or "254" for EVENT ("254" is possible only for TPDOs).

TIMER- possible values are 0-65,536. For values larger than 0 the CommPARAM field should be ASYNC ("255")

DRIVE_EVENT- bitwise hex value according to the drive events specified in MAN-CAN301IG object 0x2F20 – will be entered in decimal format

EVENT_GROUP- possible values are 1-17, values need to match the GMAS API documentation chapter 7.4.3 PDO mapping

Note: for RPDO in the EVENT_GROUP field the possible values are only 1-7

SUB_IDX- possible values are 1-24, relevant only for TPDO EVENT_GROUP of 7-10, 12-17 and for all
 RPDO EVENT_GROUP values. For other groups the value will be 0

UDP_MODE- possible values are: "2" for immediate UDP notification, "1" for notification after next cycle
"0" no UDP notification

Additionally there is a variable in the global parameters section:

SYNC- when choosing "SYNC" it is recommended to provide a value different than zero in this field (otherwise no TPDOs related to this mapping will be sent when all the axes are NON-NC). The values range is 0 (no SYNC messages) to 255/(cycle time[ms]), this field value indicates how many GMAS cycles will pass between SYNC to SYNC.

Note: All the fields MUST be entered in a decimal format even if inserted in hex format (Via UI)!!

```xml
<?xml version="1.0" encoding="UTF-16"?>
<RESOURCES
    NAME="MMCResources"
    TYPE="GMAS">
    <GLOBAL_PARAMS>
        .
        .
        .
    <NAME
        SYNC="500"/>
    </GLOBAL_PARAMS>
    <MOTION_DEVICES>
        <NODE
            NAME="b01"
            TYPE="ds402"
            ID="1"
            SUBTYPE="ELMO"
            NC_AXIS="0"
            MODULO="0"
            DIVIDER="0"
            DRIVE_TYPE="SimplIQ"
            HBT_SUPPORT="0"
            DOUT_SUPPORT="0" >

            <PDO
            PDO_NUM="3"
            DIRECTION="1"
            CommPARAM="255"
            TIMER="0"
            DRIVE_EVENT="16"
            EVENT_GROUP="7"
            SUB_IDX="2"
            UDP_MODE="2"/>
            </NODE>
        <NODE
            NAME="b02"
            TYPE="ds402"
            ID="2"
            SUBTYPE="ELMO"
            NC_AXIS="1"
            MODULO="0"
            DIVIDER="0"
            DRIVE_TYPE="SimplIQ"
            HBT_SUPPORT="0"
            DOUT_SUPPORT="0" >

            <PDO
            PDO_NUM="4"
            DIRECTION="1"
            CommPARAM="1"
            TIMER="0"
            DRIVE_EVENT="0"
            EVENT_GROUP="6"
            SUB_IDX="0"
            UDP_MODE="2"/>
            </NODE>
    </MOTION_DEVICES>
```

The fields in the SDO paragraph and their possible values are:

INDEX – the user will enter four digits hexadecimal number, this number will be written in the XML in a decimal format.

 SUB_INDEX- two digits hexadecimal number, will be stored in decimal format

DATA- 1,2 or 4 byte decimal data.

LENGTH- the length of the data ("1","2" or "4")

OP- 0 for pre – operational SDOs, 1 for operational SDOs

In this example an SDO will be sent to object 0x6085 sub-index 0x0 in axis b01, the message will hold the data 1234 and will be sent before the node becomes operational.

Note: all the fields MUST be entered in a decimal format even if inserted in hex format (Via UI)!!

```xml
<?xml version="1.0" encoding="UTF-16"?>
<RESOURCES
    NAME="MMCResources"
    TYPE="GMAS">
    <GLOBAL_PARAMS>
    ...
    ...
    ...
    </GLOBAL_PARAMS>
    <MOTION_DEVICES>
        <NODE
            NAME="b01"
            TYPE="ds402"
            ID="1"
            SUBTYPE="ELMO"
            NC_AXIS="0"
            MODULO="0"
            DIVIDER="0"
            DRIVE_TYPE="SimplIQ"
            HBT_SUPPORT="0"
            DOUT_SUPPORT="0" >

            <SDO
                INDEX="24709"
                SUB_INDEX="0"
                DATA="1234"
                LENGTH="2"
                OP="0" />
        </NODE>
        <NODE
            TYPE="ds402"
            ID="2"
            SUBTYPE="ELMO"
            NC_AXIS="1"
            MODULO="0"
            DIVIDER="0"
            DRIVE_TYPE="SimplIQ"
            HBT_SUPPORT="0"
            DOUT_SUPPORT="0" >
            </NODE>
    </MOTION_DEVICES>
    .
    .
    .
    <IO_DEVICES />
    <GENERAL_DEVICES />
    <VIRTUAL_OBJECTS />
    <SUBGATEWAY>
    ...
    ...
    ...
```

1.9 Ethernet/IP CPP libraries

We released the libraries for supporting Ethernet/IP under CPP. This truly eases the usage when working with Ethernet/IP devices.

There are two classes that work in conjunction, when working with Ethernet/IP:

- MMCEIPSession class - The class MMCEIPSession wraps the network communication functions as far as opening an Ethernet/IP session, creating tags and ending a session. The class includes the following methods:

API's

- o Error! Reference source not found. - Closes an EthernetIP session
- o Error! Reference source not found. - Creates a new EthernetIP session
- o Error! Reference source not found. - Kills the present EtherNETIP session
- o Error! Reference source not found. - Opens a new EIP session and defines an EIP allback function.

- CMMCEIPDataType class - The class MMCEIPDataType is responsible for holding the data, setting, and retaining data associated with a tag/variable defined on the Ethernet/IP network.
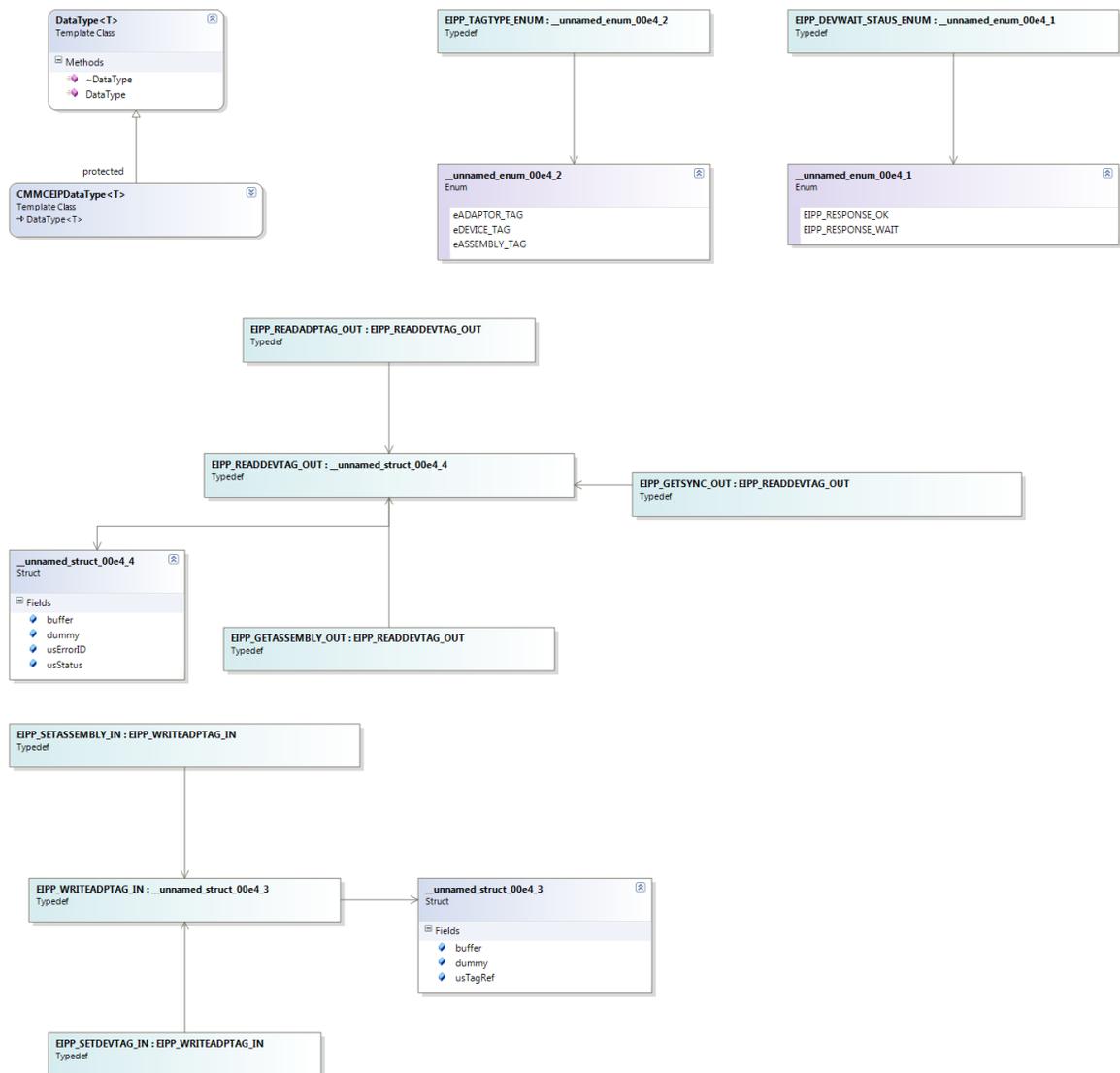
API's

- EipTagInit - Obtains a tag setting by name (*szName) or by instance (iInstance) and initializes this object with tag properties
- EipSetTag - Writes data to adapter memory
- EipGetTag - Obtains data from the adapter memory into given tData or tVar buffer
- EipCheckReply - Answers whether or not data for the last request has arrived
- EipGetData - Obtain data which has arrived asynchronously from a request

    Please refer to manual for further information.

1.10 TCP/IP and UDP CPP libraries

Historically, in order for the user to communicate to the G-MAS, only the following interfaces could be used:

- Modbus

- Ethernet/IP

If the user wished to define a propriety protocol, then managing socket experience was required. In addition, the maximum packet size, for instance, in Modbus is 256 bytes (128 registers). This was found to be limiting.

Elmo is therefore introducing two additional classes to the CPP library:

- CMMCUDP

- CMMCTCP

The user can select to use either Class, as both Classes operate in both the WIn32 and in G-MAS Programming environments. The user can also select to be a client or Server and operate the class with or without callbacks. More importantly, no socket experience is required.

The UDP includes the following methods:
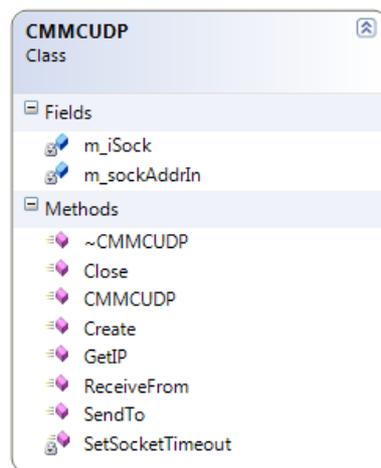
- **int Create** (unsigned short usPort, SOCK_CLBK fnClbk=NULL, int iMsgMaxSize=512);

- `int Create (unsigned short usPort, SOCK_CLBK fnClbk=NULL, int iMsgMaxSize=512);`

- `int SendTo(char* msg, short sLength)`

- `int Send (void * pData, unsigned short usSize, sockaddr_in* pSockaddr=NULL);`

- `int ReceiveFrom(char* msg, short sLength, short sTimeout) ;`

- `int Receive (void * pData, unsigned short usSize, long lDelay=0L, sockaddr_in* pSockaddr=NULL);`

- `in_addr GetIP() ;`

- `int Close();`

- `int Connect(char* szAddr, unsigned short usPort, bool& bWait, int iMsgMaxSize=512);`

Please refer to documentation for further information

In addition to the UDP and TCPIP classes, we also added a class enabling easy sending / receiving of data to the drive via EoE. This is defined with the

The following C++ functions were added (similar to Binary interpreter):

- Elmo-Get-SetSyncParam (Float + integer)

- Elmo-Get-SetSyncArray (Float + integer)

- Elmo-Get-SetASyncParam (Float + integer)

- Elmo-Get-SetASyncArray (Float + integer)

The class CMMCUDP wraps UDO implementation within the designated Operating System. The EoE class is derived from the CMMCUDP class and uses EoE communication over UDP. This section describes the public interface for CMMCUDP class.

Working with the **MMCTCP** class is similar, and has almost identical interfaces. This means, that transferring from one to another is simple.

The MMCTCP class includes the following methods:

- `int Create(unsigned int uiPort, SOCK_CLBK fnClbk=NULL, int iMsgMaxSize=512);`
- `int Accept();`
- `int Receive(int iSock, unsigned short usSize, void *pData, bool& bFail, long lDelay=0);`
- `int Send (int iSock, unsigned short usSize, void *pData, bool& bFail);`
- `int Connect(char* szAddr, unsigned short usPort, int& iSocket,bool& bWait);`
- `int Close(int iSock);`
- `int Close();`
- `bool IsWritable(int iSock);`

1.11   Enhanced Ethercat support

Major work was done in order to improve the Real-Time of the Ethercat on the G-MAS system. We

are down to jitters of 5-7us after vast intervention in the Linux Kernel.

Of course, there are limitations as far as the number of axes are concerned per cycle time.

The following guidelines apply:

-   Minimum Cycle time is 500us. 8 axes can be defined to work at this rate.
-   16 axes at 1ms. 32 axes at 2ms.
-   The minimum Mailbox time is 2ms. Mailbox time must be at least twice the cycle time.

1.12 More events in GMAS for Asynchronous Operations

Enhancements were made to time consuming API's. These API's are now ASYNCHRONOUS – meaning, they return to the user immediately, and occur in the background. When the sequence is finished, the user is notified.

Specific functions:

- ResetEx (The Reset function remained unchanged for backward compatibility).
- ChangeOperationMode.
- ConfigPDO.

The user shall receive an ASYNC reply stating which operation was over. The user can choose to receive these callbacks by setting bit 18 (Zero Based) in the event mask which is set / cleared in the following connection related functions:

- MMC_OpenUdpChannelCmd
- MMC_OpenUdpChannelCmdEx
- MMC_SetEventsMaskCmd
- MMC_ClearEventsMaskCmd

The MMC_CAN_REPLY_DATA_OUT structure was modified, and now returns information regarding the ASYNC event type.

```
typedef struct
{
        unsigned short usFunctionID ;
        unsigned short usNumerator;
        unsigned short usDatasize;
        unsigned short usPadding        ;
        unsigned short usStatus ;
        short usErrorid ;
        unsigned short usCOB_ID;
        unsigned short usAxisRef;
        unsigned char  can_data_length;
        unsigned char  data[8] ;
        unsigned char  ucAsyncEventType;
}MMC_CAN_REPLY_DATA_OUT;
```

The following enum can be returned as part of the ucAsyncEventType:

```
        eASYNC_EVENT_SDO_DOWNLOAD = 1,
        eASYNC_EVENT_SDO_UPLOAD,
        eASYNC_EVENT_CHANGE_MOTION_MODE,
        eASYNC_EVENT_INTERP_CMD_GET,
        eASYNC_EVENT_READ_DI_GROUP,
        eASYNC_EVENT_WRITE_DO_GROUP,
        eASYNC_EVENT_CONFIG_REG_PARAM_EVENT_PDO3,
        eASYNC_EVENT_CONFIG_REG_PARAM_EVENT_PDO4,
```

eASYNC_EVENT_CONFIG_USER_PARAM_EVENT_PDO3,
eASYNC_EVENT_CONFIG_USER_PARAM_EVENT_PDO4,
eASYNC_EVENT_CANCEL_PARAM_EVENT_PDO3,
eASYNC_EVENT_CANCEL_PARAM_EVENT_PDO4,
eASYNC_EVENT_RESET,
eASYNC_EVENT_INTERP_CMD_SET,
eASYNC_EVENT_SEND_RAW_DATA,
eASYNC_EVENT_EXECUTE_LABEL,
eASYNC_EVENT_CONFIG_PDO_COMM_PARAM,
eASYNC_EVENT_CONFIG_VIRTUAL_ENC,
eASYNC_EVENT_BULK_UPLOAD,
eASYNC_EVENT_SDO_DOWNLOAD_MOTION_PROCESS,
eASYNC_EVENT_SDO_UPLOAD_MOTION_PROCESS,

1.13 Enhanced Personality File

The personality now includes information regarding SDO abort reasons. The information can now be read from the standard MMC_GetErrorCodeDescriptionByID API.

A new ENUM for the abortion code (3) was added to MMC_GET_ERROR_DESCRIPTION_ENUM.

1.14 Enhanced Error Correction Support

The current implementation for the Error Correction support lacked the following:

- Only 4 Error Correction tables.
- No ability to superimpose an error correction value, for a specific axis. For instance – if axis X was corrected as a 1D Error Correction, and it was also corrected as a 2D error correction, we would not superimpose the corrections.
- The resolution (gap) must be a power of 2.
- Limited gap up to 2^{23}.

The implementation changed, WITHOUT backward compatibility:

1. Super Imposed management of the error correction. More than one table can fix a target axis.
2. Unlimited gap. Can be not only power of 2, but any floating point variable.
3. 2 Additional tables. Support up to 6 tables, not 4.(A,B,C,D,E,F tables).
4. No limitation of which axis is fixing another.
5. Start point can be double.
6. No need to handle the table offset. The GMAS now handles the search for the free memory.
7. API Change – Reference and Target axes are now not part of the API. They are part of the file.
8. Support for Auto Load of the Error Correction at G-0MAS startup.

File example

Old format:

```
Table size      64
Start offset     0
Error table dimension   2
Start position  0        0
Axis grid size  7        7
Table dimensions        8        8
Table #1 Start Data
        0       -7      -4      0       0       0       0       4
        0        0       0     -1      -4      -1      -1       0
       -1        0       0      0       0      -6       0       0
        0        0      -5      1       0      -2      -1       0
        0        0       0     -6      -2       0       0      -1
       -2       -4       0     -3       0     -20      -2       0
        0       -1       0      0     -10       0       0      -2
       -5        0       0      0       0       0       0       0
Table #1 End Data
```

New format:

```
[header]
Table size       64
Error table dimension   2
Target Axis      a01
Reference Axes   a02     a03
Start position   0       0
Axis grid size   128     128
Table dimensions         8       8
[header/]

[table]
Table #1 Start Data
        10      -70     -40     10      20      30      40      44
        30      20      10      -31     -44     -51     -61     70
        -14     50      60      40      330     -56     60      120
        560     540     -565    321     230     -432    -143    130
        0       0       0       -6      -2      0       0       -1
        -2      -4      0       -3      0       -20     -2      0
        0       -1      0       0       -10     0       0       -2
        -5      0       0       0       0       0       0       0
Table #1 End Data
[table/]
```

1.15 Additional Events to User Program

A new event was added to the GMAS. This event is called once an axis finished its initialization process. Node initialization finished event indicates whether a successful or unsuccessful node initialization has occurred.

This event supplies the user the ending state of the initialization and if an error occurred it indicates the error number.

This event will occur in the following scenarios:

- A node sent a NMT boot-up message after power on
- A node sent a heartbeat message after it was in "heartbeat error state" – a heartbeat error event occurred prior to this event.

In both cases node will be initialized by GMAS. When a node reconnects (second scenario) it will enter to error state.

The Callback Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
18	NODE_CONNECTED_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	

In addition, the **following GLOBAL ASYNC event** was added:

This event is received when an asynchronous operation that is not related to a specific node ends. This event indicates the result of this operation, on error the GMAS error code is returned and status field is different than zero, on success status and error fields are zero.

The Callback Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
19	GLOBAL_ASYNC_REPLY_EVT	Event No.	Unsigned short	0	2	
		Status	Unsigned short	8	2	
		Error ID	Short	10	2	

		Function ID	Unsigned char	12	1	

The global operations that are indicated by this event are:

- Set sync time
- Set heartbeat consumer command

The enumerators of these events can be found at MMC_events_API.h header file.

Node initialization finished event indicates whether a successful or unsuccessful node initialization has occurred.

This event supplies the user the ending state of the initialization and if an error occurred it indicates the error number.

Node initialization is performed only after a node sends boot – up message (see DS301 documentation for further details) not after every node connection on the bus. Node connected event will always take place before this event, but node initialization finished event will not always occur after node connected event.

The Callback Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
20	NODE_INIT_FINISHED_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Error id	short	10		0 - indicates successful initialization

1.16 Enhanced PDOInfo API

The MMC_GetPDOInfoCmd now returns the following data for TPDO and RPDO:

- Event group
- Communication type (sync, async, event)
- Drive event
- User array Sub index

- Timer value
- Callback mode

The structure of MMC_GETPDOINFO_OUT was modified:

```
typedef struct mmc_getpdoinfo_out
{
    int iPDOEventMode;
    unsigned int uiCommParamEventPDO;
    unsigned short usStatus;
    short usErrorID;
    unsigned short usEventTimerPDO;
    unsigned char ucRPDOCommType;
    unsigned char ucTPDOCommType;
    unsigned char ucTPDOCommEventGroup;
    unsigned char ucRPDOCommEventGroup;
    unsigned char ucSubIndexRPDO;
    unsigned char ucSubIndexTPDO;
}MMC_GETPDOINFO_OUT;
```

1.17 GeneralPDOConfiguration – now supported for DS406 devices

Users using a DS406 device, can now configure the device to return additional data via PDO 3 and PDO4 via the standard GeneralConfigPDO3/4 MMC API.

 In addition, the user can receive a callback as well on this device.

1.18 User Application Parameters Support

Today, in order to modify a specific parameter within a GMAS User Application program, the user needs to modify the code, compile and create a new executable.

For instance:

1. Default axis motion parameters

2. Communication timeouts.

3. Program behaviors. Flags, etc. …

4. And so on …

Instead of having to read hard coded constants, we want to give the user the ability to read parameters, of all types, via a dedicated API. (In the future – the user will be able to create / write to such a file).

The   *API* is to give the ability to users to read parameters from a textual based file and copy the values to local program variables.

The API is supported in the CPP Library only, currently only when working with IPC only.

An example for an XML file:

```
<?xml version="1.0" encoding="utf-8"?>
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="proposed.xsd">
        <FILE_DESCRIPTION NAME="Parameters" VERSION="NovaScan 1236" />
        <CATEGORY NAME="Profiler">
                <RESOURCES NAME="a01">
                        <AC>10000000</AC>
                        <DC>10000000</DC>
                        <JERK>1073742336</JERK>
                        <DRIVE_ID>'81'</DRIVE_ID>
                </RESOURCES>
                <RESOURCES NAME="a02">
                        <AXIS_MODE>0</AXIS_MODE>
                        <OP_MODE>8</OP_MODE>
                        <KUKU>'1073742336'</KUKU>
                        <DRIVE_ID>0</DRIVE_ID>
                </RESOURCES>
        </CATEGORY >
        <CATEGORY NAME="Communication">
                <RESOURCES NAME="a01">
                        <TIMEOUT>0</TIMEOUT>
                        <NUM_VARS>8</NUM_VARS>
                        <KUKU>'1073742336'</KUKU>
                </RESOURCES>
                <RESOURCES NAME="a02">
```

```
                    <TIMEOUT>0</TIMEOUT>
                    <NUM_VARS>8</NUM_VARS>
                    <KUKU>'1073742336'</KUKU>
              </RESOURCES>
       </CATEGORY >
       <CATEGORY NAME="Misc">
              <RESOURCES NAME="Global">
                    <DOHOMEALWAYS>0</DOHOMEALWAYS>
                    <SETPOSATTARGET>8</SETPOSATTARGET>
                    <DONOTHINGATALL>'1073742336'</DONOTHINGATALL>
              </RESOURCES>
       </CATEGORY >
</root>
```

We added the CUserParams class that includes the following methods:

- **Open** – Opens the XML file, with specific attributes – such as:
     - o   How to behave in case of not found variables,
     - o   Open file for read/write purposes.
     - o   Print to console
          - ▪   Warnings
          - ▪   Full read / write log.
- **Close** – Closes the file pointed to the XML file and release resource used for parsing the file.
- **ReadHeader** – Function that retrieves the version and file description.
- **Read** – List of overloaded function that retrieves data to given variable. In some cases also ensures that the data is not corrupt in the file by giving limitations to the read data.

Read double:
```
       int Read ( char* pCtgryVal, char* pRsrcVal, char* pTagName,
                        double &dVal,
                double      dDefault,
                double      dMin=DBL_MIN, double dMax=DBL_MAX);
```

Read long:
```
       int Read ( char* pCtgrVal, char* pRsrcVal, char* pTagName,
                long       &lVal,
                long       lDefault,
                long       lMin=LONG_MIN, long          lMax=LONG_MAX );
```

Read Boolean:
```
       int Read ( char* pCtgryVal, char* pRsrcVal, char* pTagName,
                bool       &bVal,     bool       bDefault=0);
```

Read string:
```
       int Read ( char* pCtgrVal, char* pRsrcVal, char* pTagName,
                char*      pStr, long  lLen);
```

An example for code that reads a file is as follows:

```
char * ctgrVal, rsrcVal, tagName;

ctgrVal   = "Profiler";   /* Refer to tag <CATEGORY> has value "Profiler"            */
rsrcVal   = "a01";        /* Refer to tag <RESOURCES> has value "a01"                */
tagName = "SP";           /* Refer to tag <SP> into context define by ctgrVal & rsrcVal    */
Open("kuku.xml, flags);
                          /* Look for Tag Name CATEGORY has value "Profiler",           */
                          /*      under it look for Tag Name RESOURCES have value "a01"  */
                          /*          under it look for Tag Name SP and read its value    */
Read(ctgrVal,rsrcVal,tagName,a01.m_dVelocity,100000) ;

tagName = "AC";
                          /* Read the value of tag <AC>; in CATEGORY="Profiler"/ RESOURCES="a01"
              */
Read(ctgrVal,rsrcVal,tagName,a01.m_dAcceleration,100000) ;
tagName = "DC";           /* Look for tag <DC>; CATEGORY="Profiler"/ RESOURCES="a01"  */
Read(ctgrVal,rsrcVal,tagName,a01.m_dDeceleration,100000) ;
tagName = "SP";           /* Look for tag <SP> */
Read(ctgrVal,rsrcVal,tagName,a01.m_dVelocity,100000) ;
tagName = "AC";
Read("ctgrVal,rsrcVal,tagName,a01.m_dAcceleration,100000) ;
tagName = "DC";
Read(ctgrVal,rsrcVal,tagName,a01.m_dDeceleration,100000) ;
```

Also – ReadAr for reading comma delimited arrays. Please refer to documentation for further information.

1.19 IEC – Full Implementation of G-MAS API

The following new functions were added to the IEC, and are now available in this version release:

1	MMC_ConfigGeneralRPDO3
2	MMC_ConfigGeneralRPDO4
3	MMC_CancelGeneralRPDO3
4	MMC_CancelGeneralRPDO4
5	MMC_ConfigGeneralTPDO3
6	MMC_ConfigGeneralTPDO4
7	MMC_CancelGeneralTPDO3
8	MMC_CancelGeneralTPDO4
9	MMC_NetworkInfoCmd
10	MMC_GetErrorTableStatusCmd
11	MMC_MovePolynomAbsoluteCmd
12	MMC_MovePathCmd
13	MMC_MoveLinearAdditiveCmd
14	MMC_GetGroupMembersInfo
15	MMC_GetTotalFbDepthCmd
16	MMC_AxisLink
17	MMC_AxisUnLink
18	MMC_InitTableCmd
19	MMC_LoadTableFromFileCmd
20	MMC_AppendPointsToTableCmd
21	MMC_GetTableIndexCmd
22	MMC_MoveTableCmd
23	MMC_UnloadTableCmd
24	MMC_GetLibVersion
25	MMC_GetLastError
26	MMC_GetPDOInfoCmd
27	MMC_SendSdoAsyncCmd
28	MMC_PDOGeneralReadCmd
29	MMC_PDOGeneralWriteCmd
30	MMC_GetVersionExCmd
31	MMC_GetErrorCodeDescriptionByID
32	MMC_WriteGroupOfParameters
33	MMC_GetStatusRegisterCmd
34	MMC_GetEthercatCommStatistics
35	MMC_DwellCmd

1.20 Enhanced Version Read

An important enough feature to receive its own chapter …

We now return a string of up to 120 characters long, defining the G-MAS Version, build and date.

1.21 New Kinematics in GMAS

This section describes special robot transformations of which currently Elmo supports the Delta Robot. The special transformation converts the MCS kinematic directions to ACS kinematic directions as shown below:
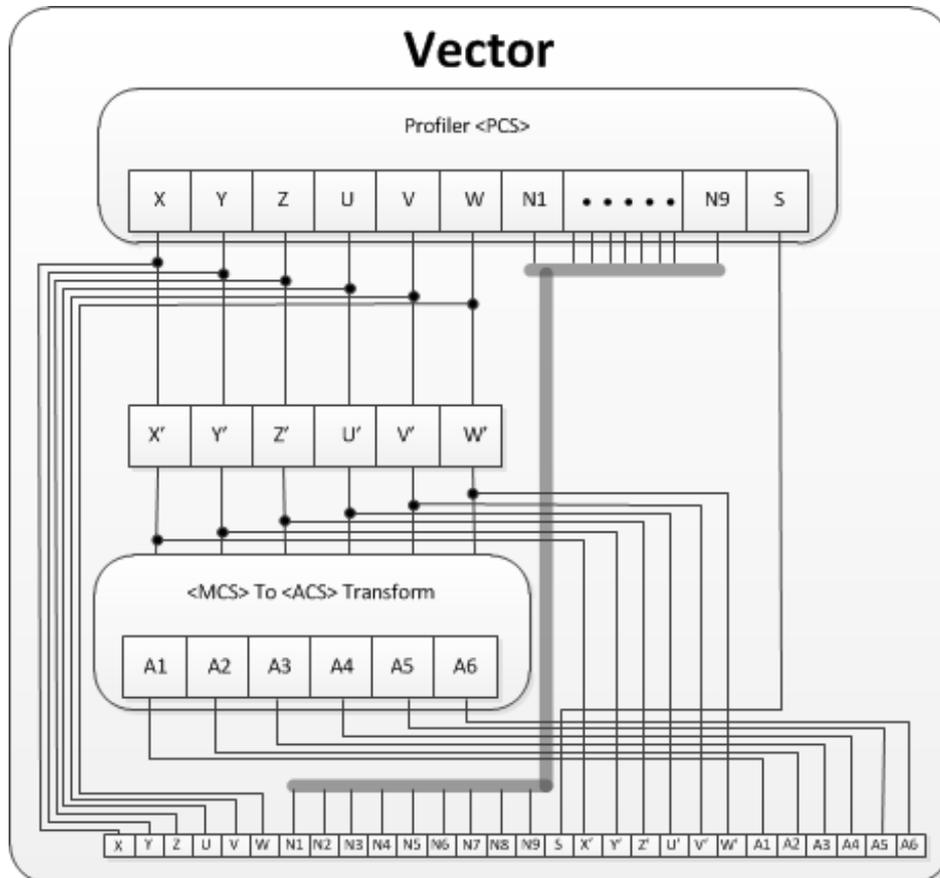
The Delta Robot is a type of Parallel Robot, with three parallel serial chains providing three degrees of freedom to the end effector. The benefits of the robot are:

- High accuracy

- High speed dynamics

- Low inertia

The Special Kinematic can be set in the G-MAS, for both MCS and PCS coordinate systems, and all types of MultiAxis motions and transitions are supported. The Limit handling mechanism also supports ACS and MCS limits.

The G-MAS can run a single Delta Robot in each vector, in up to 16 vectors, meaning that up to 16 Delta Robots can be operated at one time. Any mix of kinematic directions are supported. The units of the target position (only in the A1-A6 directions) is the same as the units of the mechanic inputs of the robot.
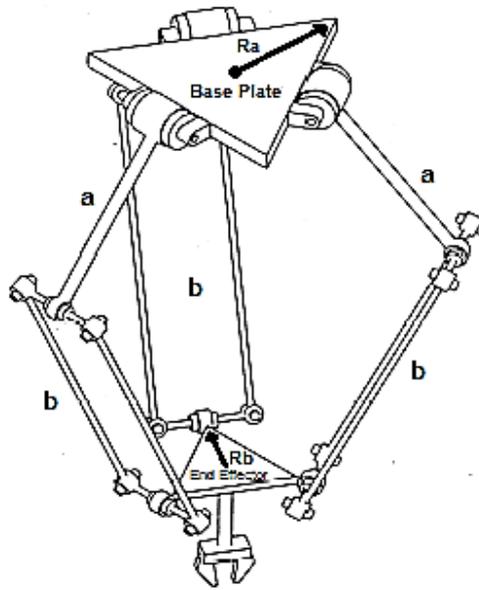
The kinematic transformations on the G-MAS convert the position and other kinematics from Cartesian space to Joint space and vice versa. The transformations can then be defined as:

- Inverse Kinematic

- Direct Kinematic

In order to clearly define the Inverse and Direct kinematics, it is necessary to define the Cartesian and Joint spaces, the origin and orientation of the Cartesian space, and the direction of the Theta in the joint space.
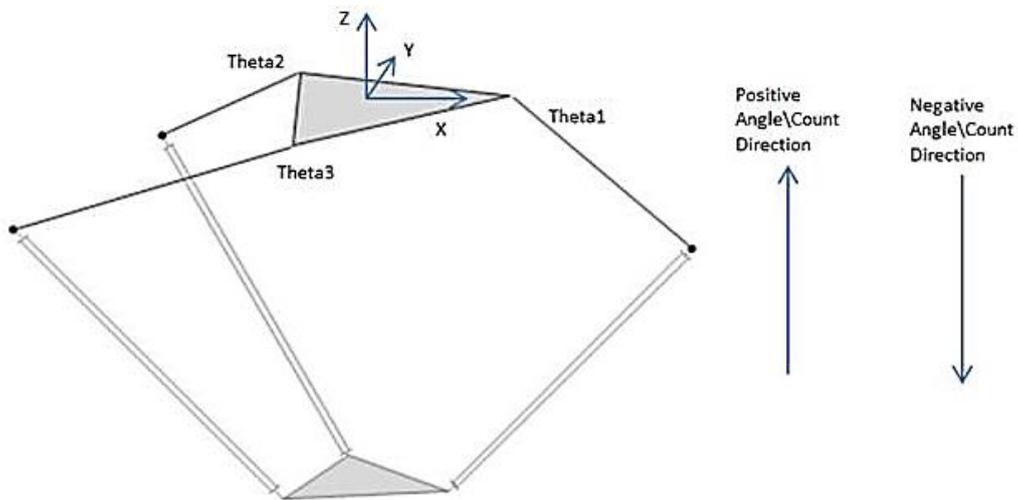
Defining Inverse and Direct Kinematics
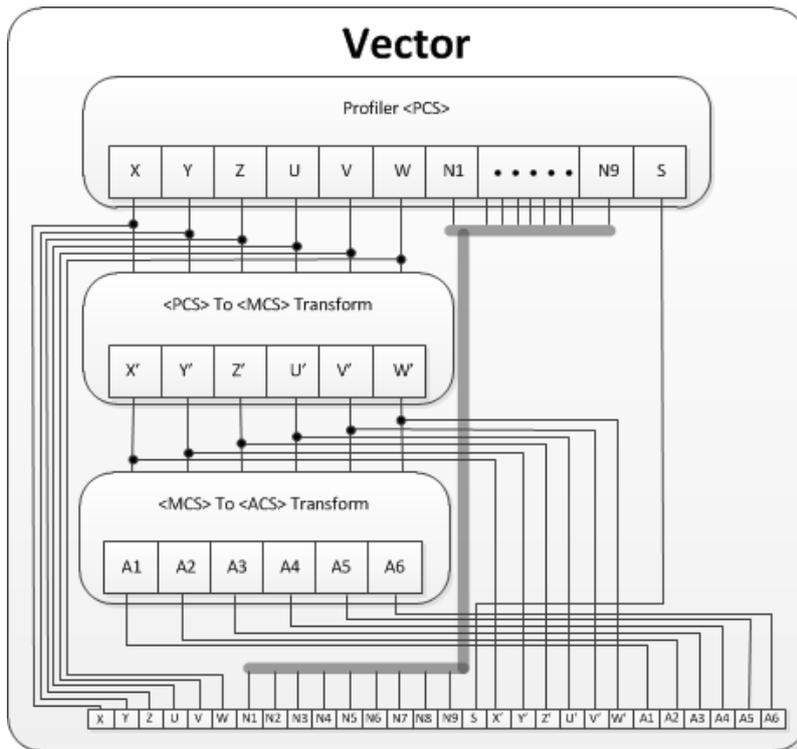
Defining Cartesian and Joint spaces

$$(X', Y', Z') \rightarrow (\Theta1, \Theta2, \Theta3)$$

$$\Theta1 = A1$$

$$\Theta2 = A2$$

$$\Theta3 = A3$$

Inverse Kinematics

Inverse kinematic can convert the target position in Cartesian space to position in Joint space of the Delta Robot. The inverse kinematic is performed in each real time cycle, during motion.
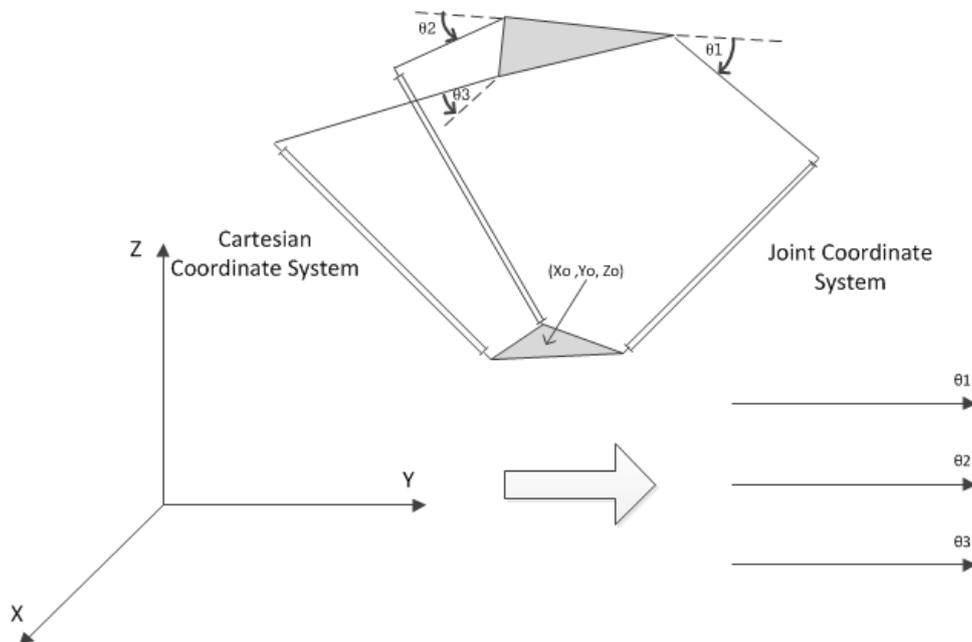
Direct Kinematics

Direct kinematics converts the target position in joint Delta Robot space to a position in Cartesian space. The transformation is performed when the actual position of the end effector is read.

Joint Coordinate System

$\theta 1$

$\theta 2$

$\theta 3$

Cartesian Coordinate System

(X_0, Y_0, Z_0)
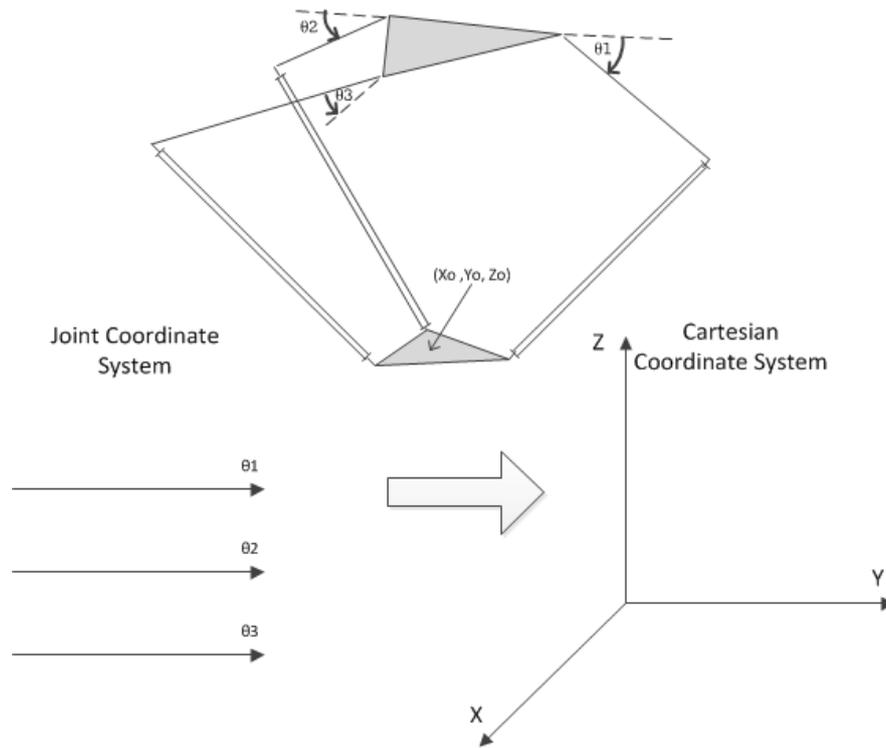
Interfaces

In order to use the Delta Robot transformation the user should define the following parameters:

- Mechanical inputs of the robot; Arm length (a), ForeArm length (b), Base Radius (Ra), and End Effector Radius (Rb)
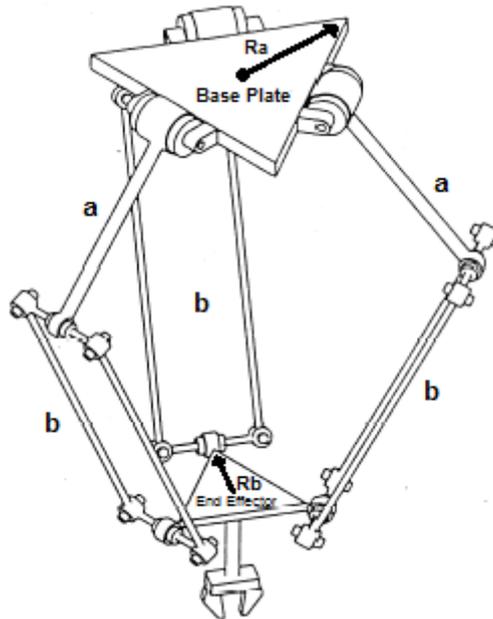
- Teaching the G-MAS the "Homing" position of the drive encoder; including the parameters Count to Angle ratio. The number of counts per one Arm revolution (2π). The count offset at zero angle.

API's

C:
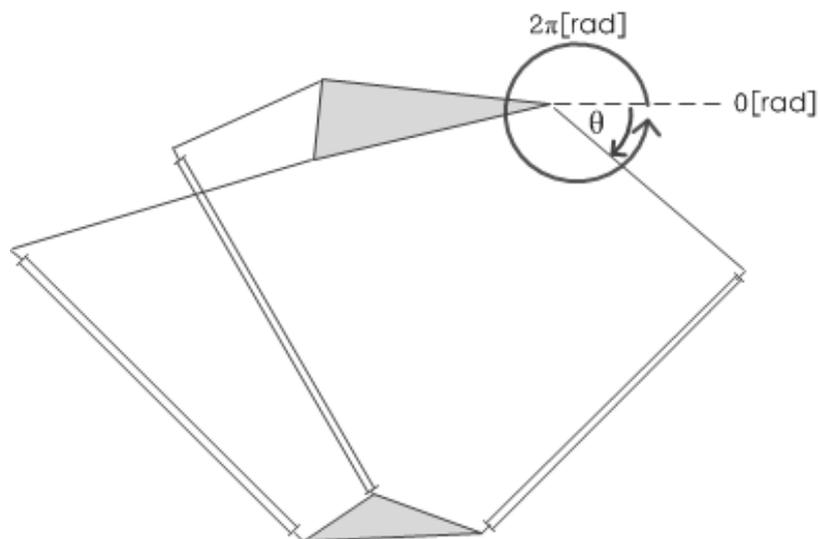
```c
int MMC_SetKinTransformEx(
            IN MMC_CONNECT_HNDL hConn,
            IN MMC_AXIS_REF_HNDL hAxisRef,
            IN MMC_SETKINTRANSFORMEX_IN* pInParam,
            OUT MMC_SETKINTRANSFORMEX_OUT* pOutParam)
```

C++:

```cpp
void SetDeltaRobotKinematics(MC_KIN_REF_DELTA stDelta) throw
(CMMCException);

typedef struct
{
        double   dbArm;
        double   dbForeArm;
        double   dbBaseRadius;
        double   dbEndEffectorRadius;
        MC_KIN_NODE_DEF sNode[NC_MAX_NUM_AXES_IN_NODE];
        int iNumAxes;
}MC_KIN_REF_DELTA;
```

Transformation between spaces - example 1

Cartesian - space
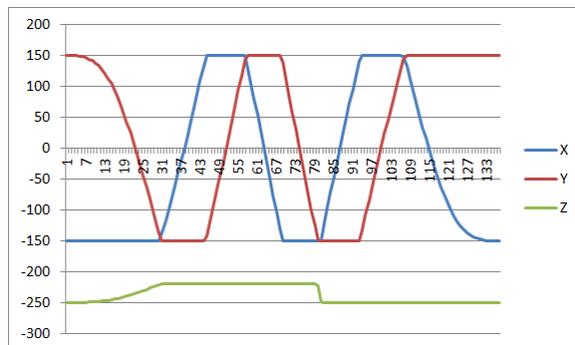
Joint - space
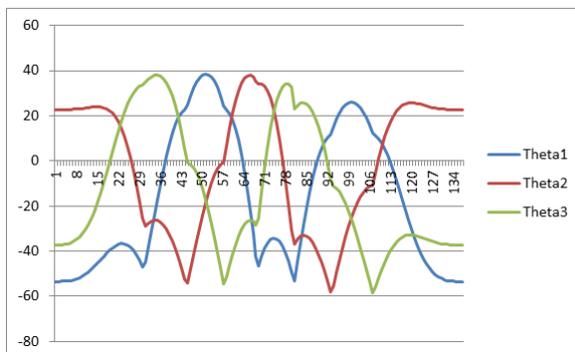
Transformation between spaces - example 2

This example shows the same motion in two different spaces, one in operational space (Cartesian) and the other in joint space (angle displacement of each leg).
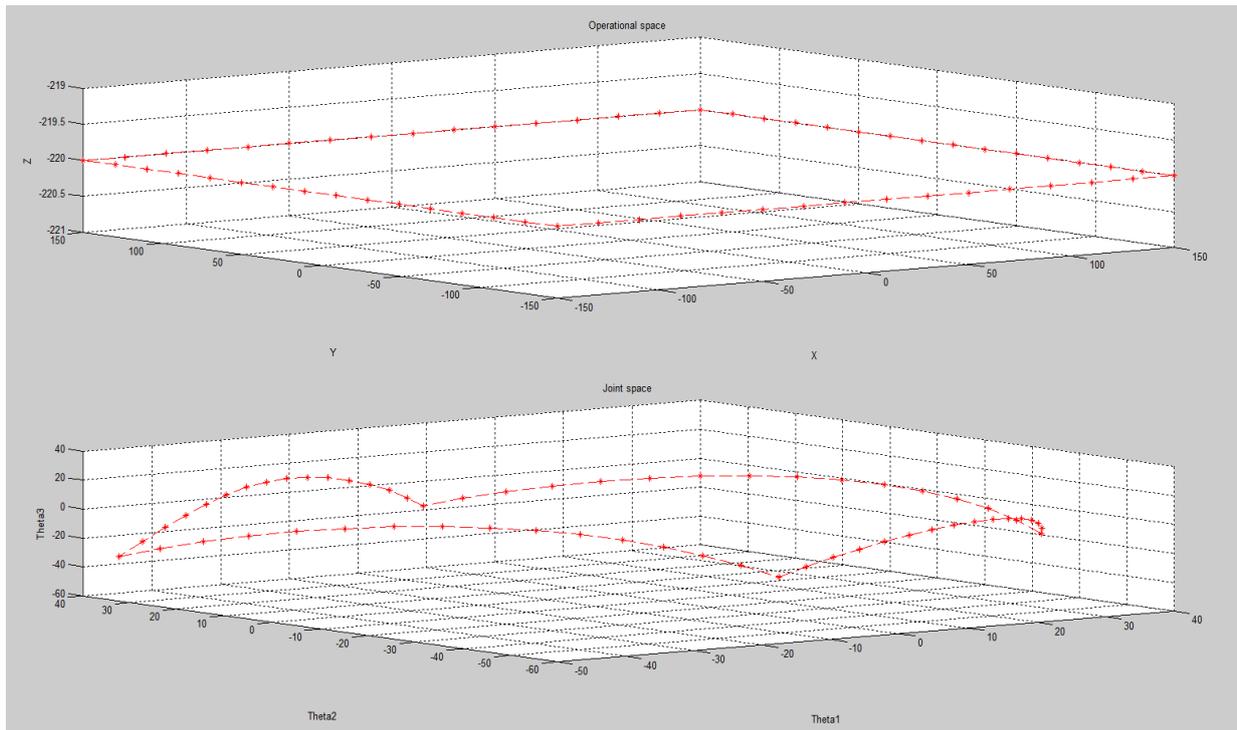
Example 1 - Set kinematic of delta robot (only 3 delta robot axes)

```
MC_KIN_REF_DELTA DeltaRobotKin;

// Set Mechanic parameters (in mm)
DeltaRobotKin.dbArm = 160;
DeltaRobotKin.dbForeArm = 320;
DeltaRobotKin.dbBaseRadius = 50;
DeltaRobotKin.dbEndEffectorRadius = 40;

// Set the number of axes
DeltaRobotKin.iNumAxes = 3;

// Assign the axes to kinematic directions

// Theta 1
DeltaRobotKin.sNode[0].eType = NC_ACS_A1_AXIS_TYPE;
DeltaRobotKin.sNode[0].hNode = AxisA.GetRef();
DeltaRobotKin.sNode[0].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC;

// Theta 2
DeltaRobotKin.sNode[1].eType = NC_ACS_A2_AXIS_TYPE;
DeltaRobotKin.sNode[1].hNode = AxisB.GetRef();
DeltaRobotKin.sNode[1].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC;

// Theta 3
DeltaRobotKin.sNode[2].eType = NC_ACS_A3_AXIS_TYPE;
DeltaRobotKin.sNode[2].hNode = AxisC.GetRef();
DeltaRobotKin.sNode[2].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC;

// Set the encoder inputs

// Encoder to Angle ratio
// (encoder with 17 bits per one revolution)
DeltaRobotKin.sNode[0].ulTrCoef[0] = 131072.0;
DeltaRobotKin.sNode[0].ulTrCoef[1] = 1.0/(131072.0);
DeltaRobotKin.sNode[1].ulTrCoef[0] = 131072.0;
DeltaRobotKin.sNode[1].ulTrCoef[1] = 1.0/(131072.0);
DeltaRobotKin.sNode[2].ulTrCoef[0] = 131072.0;
DeltaRobotKin.sNode[2].ulTrCoef[1] = 1.0/(131072.0);

// Encoder offset - the caounts value at 0 angle
DeltaRobotKin.sNode[0].ulTrCoef[2] = Theta1OffsetCnt;
DeltaRobotKin.sNode[1].ulTrCoef[2] = Theta2OffsetCnt;
DeltaRobotKin.sNode[2].ulTrCoef[2] = Theta3OffsetCnt;

GROUP.SetDeltaRobotKinematic(DeltaRobotKin);
```

Results are:
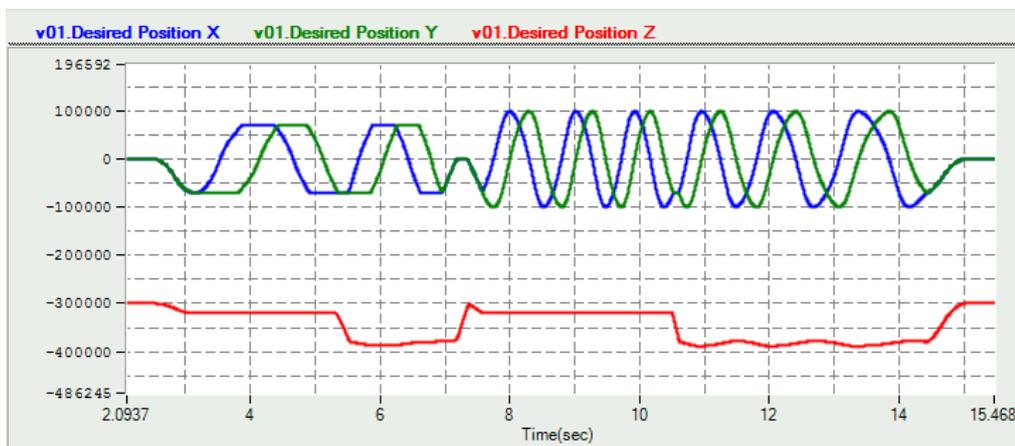
Cartesian space
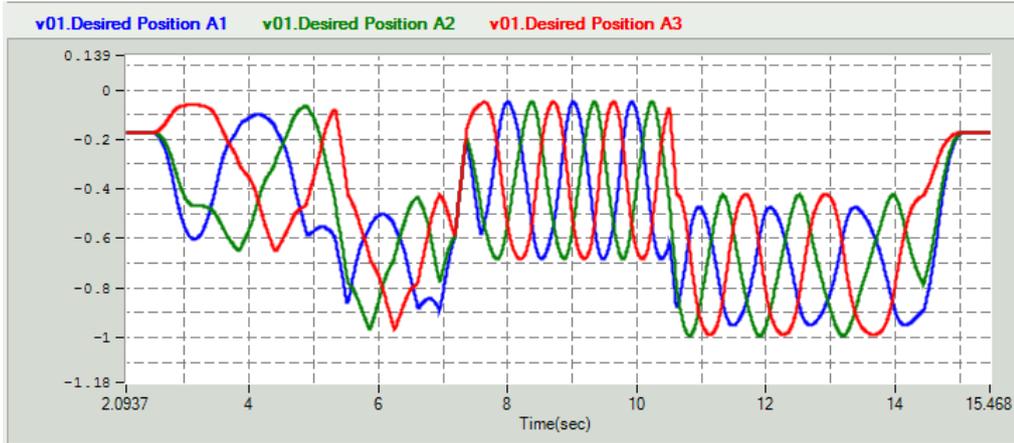
Joint space

Example 2 - Set kinematic of delta robot with one service axis

```
MC_KIN_REF_DELTA DeltaRobotKin;

// Set Mechanic parameters (in mm)
DeltaRobotKin.dbArm = 160;
DeltaRobotKin.dbForeArm = 320;
DeltaRobotKin.dbBaseRadius = 50;
DeltaRobotKin.dbEndEffectorRadius = 40;

// Set the number of axes
DeltaRobotKin.iNumAxes = 4;

// Assign the axes to kinematic directions

// Theta 1
DeltaRobotKin.sNode[0].eType = NC_ACS_A1_AXIS_TYPE;
DeltaRobotKin.sNode[0].hNode = AxisA.GetRef();
DeltaRobotKin.sNode[0].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC;

// Theta 2
DeltaRobotKin.sNode[1].eType = NC_ACS_A2_AXIS_TYPE;
DeltaRobotKin.sNode[1].hNode = AxisB.GetRef();
DeltaRobotKin.sNode[1].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC;

// Theta 3
DeltaRobotKin.sNode[2].eType = NC_ACS_A3_AXIS_TYPE;
DeltaRobotKin.sNode[2].hNode = AxisC.GetRef();
DeltaRobotKin.sNode[2].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC;

// N axis - with linear transformation
DeltaRobotKin.sNode[2].eType = NC_PROFILER_N1_AXIS_TYPE;
DeltaRobotKin.sNode[2].hNode = AxisD.GetRef();
DeltaRobotKin.sNode[2].iMcsToAcsFuncID = NC_TR_SHIFT_FUNC;

// Set the encoder inputs

// Encoder to Angle ratio
// (encoder with 17 bits per one revolution)
DeltaRobotKin.sNode[0].ulTrCoef[0] = 131072.0;
DeltaRobotKin.sNode[0].ulTrCoef[1] = 1.0/(131072.0);
DeltaRobotKin.sNode[1].ulTrCoef[0] = 131072.0;
DeltaRobotKin.sNode[1].ulTrCoef[1] = 1.0/(131072.0);
DeltaRobotKin.sNode[2].ulTrCoef[0] = 131072.0;
DeltaRobotKin.sNode[2].ulTrCoef[1] = 1.0/(131072.0);

// Encoder offset - the caounts value at 0 angle
DeltaRobotKin.sNode[0].ulTrCoef[2] = Theta1OffsetCnt;
DeltaRobotKin.sNode[1].ulTrCoef[2] = Theta2OffsetCnt;
DeltaRobotKin.sNode[2].ulTrCoef[2] = Theta3OffsetCnt;

// Set parameters of the linear transformation of the
// N axis
DeltaRobotKin.sNode[3].ulTrCoef[0] = 10.0;        // A
DeltaRobotKin.sNode[3].ulTrCoef[1] = 1.0/(10.0);// B
DeltaRobotKin.sNode[3].ulTrCoef[2] = 5;           // C

GROUP.SetDeltaRobotKinematic(DeltaRobotKin);
```

Results are:

Cartesian space

Joint space

1.22 Ethernet/IP support in IEC

This version includes Ethernet/IP support in IEC. In addition to the profiles support (Not yet supported with EAS – Basically gives the ability to map a specific tag to a parameter instead of having to call API's to do the work), we have a very basic set of API's, ready to use, with IEC.

The list of API's is as follows:

- ELMOEipOpenSession – Opens an EIP session according to XML file supplied.
- ELMOEipGetTagRef – Returns a reference to tag ((Either assembly, device or adapter tag).
- ELMOEipTag – Read / Write tag (Either assembly, device or adapter tag).
- ELMOEipTagArray -– Read / Write tag Array(Either assembly, device or adapter tag).

1.23 Async Reset Command

In addition to the MC_Reset API that already exists in the G-MAS, we added a new function by the name of: **MMC_ResetAsync.** This function is asynchronous, and returns immediately. Once the axis is initialized, the NODE_INIT event is sent to the user. This is important for systems that require fast responses whilst one of the nodes in the system is being initialized.

1.24 Additional Parameters Support in the G-MAS

The following parameters were added to the G-MAS Parameters list:

Enum	Value	Meaning
MMC_ETHERCAT_DRV_OUTPUT	78	Ethercat Drive Digital Output Parameter. Can be used instead of the MMC_WriteDigitalOutputs API. Can be used with the WriteGroupOfParameters API together with the administrative FB's.
MMC_DIGITAL_INPUT_LOGIC	79	Ability to modify the Digital Inputs logic received by the G-MAS. Mainly for 3rd party drives with no ability to change on target.
MMC_IS_FATAL_ERROR	80	Flag stating whether the G-MAS is in fatal Error.
MMC_LAST_SYSTEM_ERROR	81	Returns the last system Error
MMC_LAST_NODE_INIT_ERROR	82	Returns the last node initialization error.
MMC_LAST_SDO_ABORT_RAW_DATA	83	Returns the last SDO abort error, if it occurred.
MMC_SPEED_OVERRIDE	84	Ability to change the speed override of an axis via the parameters interface, without having to call the SetSpeedOverride API. Mainly used with the WriteGroupOfParameters API together with the administrative FB's.
MMC_FAST_REFERENCE	85	Ability to download the position (S, Actual or target) to a different drive on the network. Please refer to the Fast Reference chapter.
MMC_SET_ACDC_PARAM	86	Current AC/DC
MMC_DIGITAL_INPUT_PARAM	87	Returns the Drives Digital Input. This gives the ability to use the WaitOnConditionFB with this parameter, and mask specific bits.
MMC_CAN_DRV_OUTPUT	88	Same as *MMC_ETHERCAT_DRV_OUTPUT*, but for CAN.
MMC_DS402_CONTROL_WORD	89	Read only Control Word value
MMC_DS402_STATUS_WORD	90	Read only Status Word value
MMC_STATUS_REGISTER	91	Read Only Status Register. Value returned in the MMC_GetStatusRegisterCmd API.
MMC_MCS_LIMIT_REGISTER	92	Read Only Limit Register. Value returned in the MMC_GetStatusRegisterCmd API.

1.25 Fast Reference Support to Drive

A new capability in the G-MAS, is the ability to send the position of an axis to a different axis on the Ethercat Network. One of the following can be sent:

- Actual Position.
- Target Position
- 'S' – Position along the path (Group axis can send the position along the path to a specific axis, used mainly for >2D compare abilities).

The data is inserted to the FAST_REFERENCE 0x2005 in the drive. What does the drive do with it ? it all depends on the implementation. It can be mapped to a socket, and then used in a gain scheduling table. It can be used as an auxiliary position reference, etc …

In order to work with the FAST_REFERENCE, it must be of course mapped via the Ethercat Configurator.

The User must define how he wishes to work: Which axis Reference to download the data to and which parameter to map. This behavior is changed by accessing the MMC_FAST_REFERENCE parameter in the parameters list, where:

The 16 MSb of the parameter hold the AxisReference of the target. The LSb can hold one of the following:

1 - TargetPos

2 - ActualPos

3 - S Position along the path.

1.26 Bug Fixes in this Version

The following lists the bug fixes in this version. For more information, please contact us:

2693	1.1.1.8B1
2692	1.1.1.8B1
2127	1.1.1.7
1975	1.1.1.6
144	1.1.1.7
148	1.1.1.6
215	1.1.1.5
214	1.1.1.5
202	1.1.1.6
747	1.1.1.5
786	1.1.1.6
1249	1.1.1.7
1329	1.1.1.8B1
1324	1.1.1.8B1
1459	1.1.1.7
1490	1.1.1.6
1495	1.1.1.7
1499	1.1.1.6
1531	1.1.1.6
1652	1.1.1.7
1654	1.1.1.6
1674	1.1.1.7
1753	1.1.1.6
1752	1.1.1.6
1751	1.1.1.7
1744	1.1.1.8B1
1728	1.1.1.6
1798	1.1.1.6
1794	1.1.1.6
1787	1.1.1.7
1765	1.1.1.5
1762	1.1.1.5
1820	1.1.1.6
1828	1.1.1.6
1834	1.1.1.7
1839	1.1.1.7
1840	1.1.1.5
1843	1.1.1.7
1844	1.1.1.7
1845	1.1.1.7
1846	1.1.1.7
1874	1.1.1.6
1873	1.1.1.6
1871	1.1.1.6
1870	1.1.1.6
1924	1.1.1.5
1920	1.1.1.5
1919	1.1.1.8B1
1916	1.1.1.5
1915	1.1.1.5
1912	1.1.1.5
1925	1.1.1.5
1938	1.1.1.5
1940	1.1.1.5
1942	1.1.1.5
1943	1.1.1.6

1944	1.1.1.5
1952	1.1.1.6
1958	1.1.1.6
1965	1.1.1.6
1967	1.1.1.6
1970	1.1.1.6
1971	1.1.1.7
1972	1.1.1.6
1977	1.1.1.6
1978	1.1.1.6
1983	1.1.1.6
1984	1.1.1.6
2012	1.1.1.6
2017	1.1.1.6
2022	1.1.1.6
2024	1.1.1.7
2033	1.1.1.6
2034	1.1.1.6
2035	1.1.1.6
2109	1.1.1.7
2134	1.1.1.7
2150	1.1.1.7
2265	1.1.1.7
2352	1.1.1.7
2354	1.1.1.7
2357	1.1.1.7
2469	1.1.1.7
2470	1.1.1.7
2471	1.1.1.7
2473	1.1.1.7
2482	1.1.1.7
2512	1.1.1.8B1
2569	1.1.1.8B1
2700	1.1.1.8B1
2703	1.1.1.8B1
2782	1.1.1.8B1
1930	1.1.1.5
1909	1.1.1.5
1913	1.1.1.5
1869	1.1.1.6
1726	1.1.1.6
1534	1.1.1.5
1212	1.1.1.7
234	1.1.1.5
231	1.1.1.5
141	1.1.1.5

Release Notes - New Firmware Version for GMAS

Version 1.1.1.4 and Library Update 232

1. General

The "*uImage_v1.1.1.4_2012_10_02.gms*" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. NetworkInfo shall be updated once heartbeat on axis is thrown and shall reflect the true state of the device.
2. Event once an axis returns to network (once boot up is sent from device – user will receive notification).
3. Error State cancellation for DS401, DS406, DS301 option as per parameter that must be set.
4. Enhanced FULL Emergency message including the Vendor, Error Code, Error ID.
5. Event to user once Bin Interpreter or SDO reply arrived.
6. Spline support for zero speed segments, on PT Splines.
7. Modulus support for Virtual Encoder following a virtual axis with positions > 32bit.
8. PDO and SDO initializations can be done from GMAS resource file (Still … no support in EAS), in devices pre-operational mode.
9. Updated functions for setting and reading the digital output parameter in the CPP library.
10. Ability to bind a specific port in CMMCUDP CPP class to listen / send messages on in able to promise that the GMAS will always use the same port in the user program.

1.1 NetworkInfo Reflecting true status of CANopen device

The NetworkInfo structure that is returned as part of the MMC_NetworkInfoCmd function now updates whether the axis 'disappeared' from the network. This feature, is of course available only in conjunction with the Heartbeat mechanism. If the GMAS ceased to receive the heartbeat from the device, the NetworkInfo structure shall reflect this. If the Heartbeat mechanism is not initialized, the NetworkInfo structure will not be updated.

1.2 New Notification when device is powered on

A new event type is now defined:

#define NODE_CONNECTED_EVT   18

This event is called once the user configured the event bit (bit 18, 0 based) to return an event. Once this bit is set (in the MMC_OpenUdpChannelCmd or MMC_SetEventsMaskCmd functions) an event is triggered to the user. Please refer to User manual for specific byte definition.

1.3 Error State cancellation for DS401, DS406, DS301 devices

There are CANopen devices (DS401, DS406, DS301) that send **emergency warning** messages. In this case, the GMAS enters the ERROR_STATE – only to be exited by calling the MMC_Reset function. In order not to enter the ERROR state when an emergency is sent by the device, a parameter can be set. This parameter can be called via the user program, or can be saved to flash.

The enumeration of the parameter is: EMCY_SET_ERR and can be found in the MMC_PARAMETER_LIST_ENUM group (MMC_general_API.h).

The parameter can be read/set by calling the MMC_ReadBoolParameter / MMC_WriteBoolParameter functions.

Setting the parameter to 0 will cancel the entrance to Error State. the default value is 1.

1.4 Enhanced FULL emergency message

There is now support for additional information in the emergency callback. Now the callback also includes the error register and the manufacturer specific error code. Please refer to User manual for specific byte definition.

Notes:

- Please follow the size of the callback message returned in order to determine if supported in the version you are using.
- This is not yet supported for Ethercat.

1.5 Event to user once Bin Interpreter or SDO reply arrived

Users who wish to work in an event-driven environment can now send SDO's and Binary interpreter (CAN Only) and receive an event once a reply arrived to the GMAS.

This event is called once the user configured the event bit (bit 19, 0 based) to return an event. Once this bit is set (in the MMC_OpenUdpChannelCmd or MMC_SetEventsMaskCmd functions) an event is triggered to the user. Please refer to User manual for specific byte definition.

1.6 Spline Support for Zero Length Segments

There is now support for spline files including Zero length segments. This is supported for Position + Time (PT) based Spline files such as the following Spline file:

```
Spline Mode:     1
Spline dimension:        3
Number of spline points:          991
Spline data start
        0.03333         76.200  76.200  0.000
        0.03333         76.200  76.200  0.000
        0.03333         76.200  76.200  0.000
        0.03333         76.200  76.200  0.000
        0.03333         76.200  76.200  0.000
        0.03333         76.200  76.200  0.000
        0.03333         76.200  76.200  0.000
        0.03333         76.200  76.200  0.000
        0.03333         76.200  76.200  0.000


        …
```

1.7 FULL Modulus support for Virtual Encoders

The virtual encoders now support the following of a virtual axis where the position of the virtual axis

> 32bit (MAX_LONG).

The user can now define a modulus value on the virtual encoder, while following a virtual axis whose value is larger than MAX_LONG.

1.8 GMAS Resource file – now a CAN initialization resource file

The resource file of the GMAS now includes CAN PDO 3 – 4 initializations, SDO downloads – all in the

CANopen preoperational mode.

This is exceptionally good for novices who find it difficult using the GMAS APIs for initializing PDO and SDO's. There is still no EAS support for this feature yet.

1.9 New overloaded CPP functions for Digital Output handling

There are new CPP functions for handling digital outputs (irrelevant if it is CAN or Ethercat)

1.1 Binding of UDP Ports to specific GMAS port in MMCUDP CPP Class.

The UDP class we have today, is more for EoE support, and was designed as such. Modifications were made on this class, where the user can define the port the GMAS is to listen on / send data from.

The Create function now includes an additional, optional, parameter – iGMASUDPPort. The default is 0.

m_udp.Create("192.168.1.33",6000, **6666**) ;

So, in the line above – a socket is created to send data to 192.168.1.3. Data will be sent to port 6000 from port 6666 (or / and will listen for data on port 6666).

In future versions, there is to be support for FULL TCPIP/UDP client/server functionalities.

Release Notes - New Firmware Version for GMAS

Version 1.1.1.1 and Library Update 229

1. General

The "*uImage_v1.1.1.1_2012_07_22.gms*" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. Numerous bug fixes around closing-reopening the Multiaxis process. Waiting for the LPT before exiting.
2. PVT bug fixes.
3. 0x6071 is downloaded (= 0) via SDO only in case we exit from CYCLIC_TORQUE during DS402 change mode.
4. CPP support for PVT.

Release Notes - New Firmware Version for GMAS

Version 1.1.1.0 and Library Update 228

2. General

The "*uImage_v1.1.1.0_ .gms*" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

5. New Ethercat Supported Features
    - Cyclic Velocity Support.
    - Cyclic Torque – Debug support.
    - New Real Time PDO variables were added
6. Motion Limits Handling Support in GMAS.
7. CAN Bulk Upload handling.
8. S Spatial path.
9. 6th order Polynomial Support for Corner Deviation Transitions
10. New List of parameters that can be read.
11. New Personality.
    - Additional recording parameters.
    - Recording parameters as per group.
    - Drive emergencies.
12. PVT Support in GMAS.
13. Ability to read emergencies reason from drive API.
14. Improved functions:
    - mmc_enablemotionendedevent.
    - mmc_disablemotionendedevent
15. Changes related to Parameters file
    - When the user sets a parameter out of limit – the default value is set.
    - Parameter is inserted to the file according to type.
    - Version is added to the parameters file.
    - RAM Snapshot creation – The user may now upload a memory snapshot of the GMAS parameters.

16. New error codes related to :
    - o SW/HW limits
    - o Bulk Upload
    - o PVT
17. New variable + API to read the new StatusRegister variable and Data recording support for the new status register.
18. ACStoMCS and MCStoACS improvements.
19. Special Assign and Special retrieve functionality parameters.
20. Give FB's additional properties:
    - o Check Limits,
    - o Allow motion when in limit.

21. Reduced cycle in Ethercat data recording due to code optimizations.
22. IEC programming enhancements.
23. GMAS Home on block support
24. Enhanced and New 'C' and CPP API's
    - o CPP library
        - ▪ New CPP overloaded functions for Single / Group motions
        - ▪ Default constructor value for eDirection parameter in the MC_MOTIONPARAMS_SINGLE class.
        - ▪ New CMMCMotionAxis class for mutual Single and Group motion activities such as ECAM, PVT.
        - ▪ New CMMCPVT class supporting PVT functionality – Such as LoadTable, InitTable, AppendPoints.
        - ▪ Bug fix in CMMCPPGlobal class – the Singleton class did not properly release the instance. The deallocation of the instance caused a stack overflow.
        - ▪ CMMCMessage is not a member of CMMCPPGlobal class in case of win32. This means that currently, obtaining the reason for the last error is not available via win32.
    - o C library
        - ▪ All new Error codes were updated in the MMC_Definitions.h file.

```
#define NC_HIGH_LOW_SW_LIMIT_CONTRADICTION          (-211)
#define NC_MOTION_FORBIDDEN_ON_HW_LIMIT             (-212)
#define NC_MOTION_FORBIDDEN_ON_ACS_SW_LIMIT         (-213)
#define NC_MA_FORBIDDEN_DIRECTION_ON_ACS_LIMIT      (-214)
#define NC_MOTION_FORBIDDEN_ON_MCS_SW_LIMIT         (-215)
#define NC_SA_MOTION_TOWARD_SW_LIMIT_FORBIDDEN      (-216)
#define NC_MA_MOTION_TOWARD_SW_LIMIT_FORBIDDEN      (-217)
#define NC_SA_FORBIDDEN_DIRECTION_ON_LIMIT          (-218)
#define NC_MA_FORBIDDEN_DIRECTION_ON_MCS_LIMIT      (-219)
#define NC_BULK_UPLOAD_SIZE_DOES_NOT_MATCH          (-220)
#define NC_BULK_UPLOAD_SEND_ACK_FAILED              (-221)
#define NC_PVT_ECAM_FAILED_TO_OPEN_FILE             (-222)
#define NC_PVT_ECAM_FAILED_TO_PARSE_HEADER          (-223)
#define NC_PVT_ECAM_UNCOMPATIBLE_TABLE_MODE         (-224)
#define NC_PVT_ECAM_FAILED_TO_PARSE_DATA            (-225)
#define NC_PVT_ECAM_TABLE_ALLOC_FAILED              (-226)
#define NC_PVT_ECAM_DIMENSION_DOES_NOT_MATCH        (-227)
#define NC_PVT_ECAM_MEM_HANDLE_OUT_OF_RANGE         (-228)
#define NC_PVT_ECAM_UNALLOCATED_SEGMENT_ERROR       (-229)
#define NC_PVT_ECAM_NUM_OF_PTS_DOES_NOT_MATCH       (-230)
#define NC_PVT_ECAM_PATH_FOR_ANOTHER_AXIS           (-231)
#define NC_PVT_ECAM_AT_LEAST_2_PTS_REQUIRED         (-232)
#define NC_PVT_ECAM_JOURNAL_HANDLE_OUT_OF_RANGE     (-233)
#define NC_PVT_ECAM_INVALID_FILE_NAME               (-234)
#define NC_BUFFER_INSERTION_FORBIDDEN_ON_LIMIT      (-235)
#define NC_TORQUE_OUT_OF_RANGE                      (-236)
#define NC_PARAMETER_INDEX_OUT_OF_RANGE             (-237)
#define NC_CANNOT_DEFINE_SAME_NODE_TWICE            (-238)
#define NC_PVT_ECAM_INDEX_RETRIEVE_FAIL             (-239)
#define NC_PVT_ECAM_INDEX_IS_CLOSE_TO_CURRENT       (-240)
#define NC_PVT_ECAM_TABLE_IS_NOT_INITIALIZED        (-241)
```

```
#define NC_PVT_ECAM_OVERFLOW_TABLE_ERROR           (-242)
#define NC_PVT_ECAM_START_INDEX_OUT_OF_RANGE       (-243)
#define NC_PVT_ECAM_END_INDEX_OUT_OF_RANGE         (-244)
#define NC_PVT_ECAM_NO_POINTS_IN_TABLE             (-245)
#define NC_PVT_ECAM_NOT_ADJACENT_APPEND            (-246)
#define NC_PVT_ECAM_UNDERFLOW_THRESHOLD_TOO_BIG    (-247)
```

- Updated recording parameters: NC_REC_PARAM_NAMES_ENUM
- New parameters were updated in the MMC_PARAMETER_LIST_ENUM

```
MC_MAX_CURRENT_PARAM                = 56,
MMC_LIMIT_STOP_DECELERATION         = 57,
MMC_LIMIT_STOP_JERK                 = 58,
MMC_SET_VECTOR_VELOCITY_PARAM       = 59,
MMC_MCS_SW_LIMIT_LOW_POS_ARRAY      = 60,
MMC_MCS_SW_LIMIT_HIGH_POS_ARRAY     = 61,
MMC_MCS_S_DIRECTION                 = 62
```

- New functions:
  - **GetErrorCodeDescriptionByID** – A new function that returns a textual description and resolution for an error or drive emergency.
  - **GetGroupMembersInfo** – A new function that returns information (ID, name, axis references) for all physical axes within a Group axis.
  - **GetStatusRegisterCmd** – A new function that returns the Status register of the axis. The Status register currently holds information regarding the HW and SW limits of the Single / Group axis.
  - **StartBulkUploadCmd** – Starts the CAN Bulk upload sequence.
  - **GetBulkUploadStatusCmd** – Retrieves the CAN Bulk Upload status.
  - **GetBulkUploadDataCmd –** Retrieves the Bulk Upload data.
  - **LoadTableFromFileCmd** – Loads a PVT / ECAM file.
  - **AppendPointsToTableCmd** – Append user points to array.
  - **InitTableCmd** - Initialize PVT / ECAM table.
  - **MoveTableCmd** – Start a PVT / ECAM motion based on the points loaded.
  - **UnloadTableCmd** – Unload PVT / ECAM table.

25. Bug Fixes:
   - Wrong place of parameter updating "ucIsArcInserted"
   - Ability to work with N axes move linear relative function
   - Ability to work with N axes using Circle mode "Angle".
   - XQ binary interpreter bug fix. Additional UDP message was sent even though not necessary.
   - Referencing released pointer in reactor bug fix.
   - The rounding of number(iPos) caused an offset of 1 in case of negative number.
   - Triggering Actual Current is now available (but its units are in mAmps when being triggered).

2.1 New Ethercat Supported Features

The GMAS now also support the Cyclic Velocity and Cyclic Torque DS402 operation modes.

Cyclic synchronous torque mode overview   Cyclic synchronous velocity mode overview
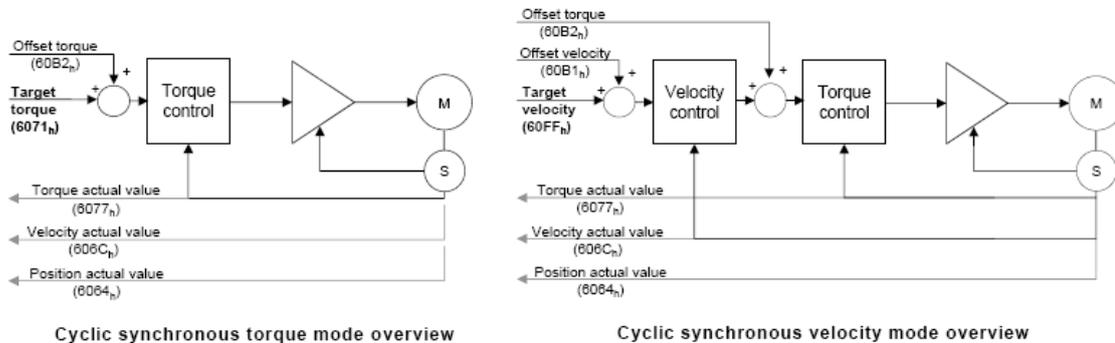
New Ethercat Supported Features

- o Cyclic Velocity Support. This mode is obtainable by changing the Operation mode to *Cyclic Velocity*. The relevant map able DS402 objects must be pre-initialized (Object 60FF - Target velocity Commands). When calling the *MoveVelocity* API and the axis is in *cyclic velocity* operation mode, the 60ff object is updated according to speed sent in motion profile.
- o Cyclic Torque – Debug support. This is not the full feature to be supported by the GMAS, but only a debug mode. This mode is obtainable by changing the Operation mode to *Cyclic Torque*. The relevant map able DS402 objects must be pre-initialized (Object 6071 - Target torque Commands). The mode is currently supported by calling the MoveAbsolute / Relative / Additive function blocks. The rated torque is sent, so it is under the user's responsibility to keep the rated current within permitted limits.
- o New Real Time PDO variables were added. Object 60B1 – Velocity Offset: With this new object mapping, the user can now benefit from better motion interpolation algorithm within the drive when working in Cyclic Position Mode. This allows smoother motions, and clean acceleration (current) feed-forward gains to be used inside the drive for ultimate servo and motion performances.
- o Object 60B2 – Torque Offset: Using this object mapping, the user can now on-the-fly change max torque limitations within the drive. This can be used for wire-bonding applications for example, where at certain motion segments, limited torque motion is required. Object 6071 – Target Torque and Object 60FF - Target velocity Commands: These objects are used for the Cyclic Velocity and Torque modes

2.2   Motion Limits Handling Support in GMAS

The GMAS now fully handles both Software and Hardware Motion limits. This important new feature, further improves the application protection capabilities when working with the GMAS as a Multi-Axes controller.

The Limits Handling mechanism is implemented in a way that can prevent illegal motion insertion. This means that if the user tries to command a motion (in either position or velocity modes), which is beyond the allowed motion limits, the motion will be rejected with an appropriate error code.

This new important feature applies to:

- Single Axis Motions: NC and Distributed motions.
- Group Axes Motions:
    - ACS – Axis coordinate system.
    - MCS – Machine coordinate system (Cartesian X-Y-Z …).

New Axis statuses were added, for Single axis and Groups, to report to the user the exact axis limits status.

All limits statuses are available in the "Bulk Read" functionality to allow fast and immediate information upload of around 30 axes in a single TCP-IP packet (less than 0.5 milliseconds). User can register Events to user programs when an axis is stopped due to limit.

Please refer to full documentation regarding this feature.

1.1 CAN Bulk Upload Support

An improved bulk-mode upload data recording format has been developed and added to the Gold Drives and GMAS CANOpen protocol.
This is an enhancement to the standard DS-301 Segmented SDO upload. The CAN Upload mode accelerates data uploading from the gold drive by factor of ~x10 compared to the standard Segmented SDO upload mechanism. This is mainly used for fast Data Recording upload process (used by the recorder), and in the future will also support the Live-Scope capability in Drive level.

The following benchmark table describes the expected uploading time of a 64 kB buffer from the drive to the PC using the GMAS CAN Bulk Upload mechanism

CAN Baud rate	Bus load (CAN)	Bus load (TCP-IP)	Total Communication Time
125 Kbit/s	31%	55%	28.47 sec
250 Kbit/s	44%	38%	10.4 sec
500 Kbit/s	56%	25%	4.3 sec
1 Mbit/s	76%	12%	1.443 sec

Please refer to full documentation regarding this feature.

1.1 2D and 3D Spatial Trigger Generation

The GMAS can now export its internal spatial 2D or 3D profile position and velocity information to a drive on the network, using the standard Target Position (607A) and Velocity Offset (60B1) objects.

The spatial position profile, designated as "S" group member within the GMAS, represents the distance along the path of 2D and 3D spatial motions. It starts from "0" at the beginning of any new Group) vector motions, and is accumulated over the spatial motion profile (over all motion segments or function blocks). The 'S' member must be 'mapped' (using the SetKinematic function), and an MCS motion function is to be called.

Specific new Reset function is available to clear the value of "S" if needed. This is called by a standard Group Parameter assignment (SetParameter CPP function) to the MMC_MCS_S_DIRECTION index (62) within the MMC_PARAMETER_LIST_ENUM list.

The main usage this feature is to support spatial 2D or 3D position based triggers to laser or other measuring equipment.

In the example below, the 'S' direction is mapped to'b01' axis. Full correlation can be found between the S desired position and b01 desired position.

The same applies to Velocity and AC/DC parameters.
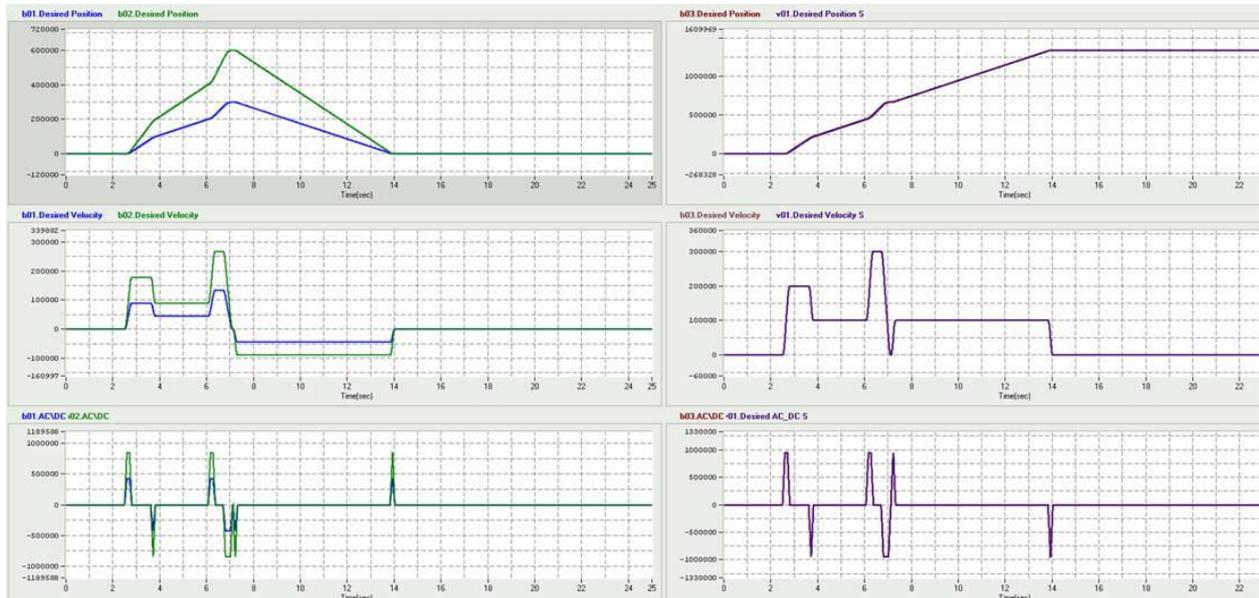
1.2 GMAS Personality Data Base Support

The GMAS can now export its own Personality Data Base file to users. The EAS (Elmo Application Studio) is using products Personality Data Base to match tools behavior to specific target versions.

Users, in addition can now benefit from the advanced error handling and diagnostic reports that are also included, to trap and report system, network and drive error codes and descriptions.

The following information is now coded in the GMAS personality Data Base:

-     Recording vectors per category.
-     FULL GMAS parameter list and description.
-     GMAS Error codes with descriptions and resolutions.
-     Drive Emergency Codes

1.3 6th order Polynomial Spline Support for Corner Deviation Transitions

In order To the reach variety of 2D and 3D motion options supported by the GMAS, we have now added a 6th order polynomial "Corner Deviation" Transition mode that allows completely smooth motion over line to line and Circle to line motion segments.
The new mode of operation is a completion to the already available 5th order spline algorithm that is used for other transition modes. With this new option, all GMAS motion and transition mode options guarantee smooth velocities and acceleration over all motion profiles, for better overall motion
performances.

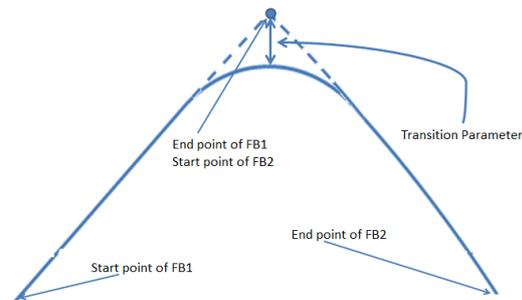## 1.4 New Motion Modes for PVT

The GMAS now fully supports both Spline (arbitrary Position Table Points), and PVT (Position/Velocity/Time) motion profiles.

- o Splines can be used when a position motion profile is known in advance, but without any velocity or time information. The GMAS will calculate motion speeds with the given system constraints, and will execute a smooth motion profile (position / velocity / acceleration are continuous), with a cubic polynomial interpolation method.
  - Spline motion is supported for groups, with up to 16 axes.
- o PVT motion can be used when the user has on the fly motion profiles changes, and when both position and velocity are specified. The GMAS will interpolate a cubic polynomial motion profile for each 3 input segments. All The PVT profile information can be given in full double precision floating point formats.
  - PVT is supported for both Single and Group axes (in both ACS and MCS modes).
  - Both Absolute and Relative positions and timing can be defined.
  - Large cyclic (option) PVT memory cyclic buffers:
    - Single axis PVT motion buffer > 40K points (of motion segments!)
    - 3 Axes PVT buffer size > 17K motion segments.
    - 16 Axes PVT buffer size > 3K motion segments.
  - PVT Motion Modes:
    - Static mode – Based on a file or table points.
    - Dynamic mode – Based on table points, sent on the fly to GMAS. Head / Tail event mechanism is available for this operation mode, for practically infinite motion profiles execution.

Please refer to the full PVT documentation for further reference on this mode.

1.5 IEC Features and Modifications

The following new enhancements are now included within the GMAS IEC programming tools:

- Ability to compile the IEC to Released - 'C' code:
  - o Much faster execution for release code.
  - o This feature will be supported by the EAS IEC environment in the near future
- Profile – Ability to 'map' an IEC variable to following:
  - o IO – (DS401, DriveIO).
  - o Modbus parameters.
- On the fly change of image:
  - o This powerful feature allows the user to change their code while system is running, maintain global variable values and code flow.
  - o This feature will be supported by the EAS IEC environment in the near future.
- Both Cold and Warm Starts are now supported

Changes were performed to the existing Function blocks. There should be no issue of backward compatibilities here.

DONE bit was removed, and instead a valid bit was added (in the location of the previously existing DONE bit) from the following Function Blocks / Functions:

- o MC_ECATIOReadAnalogInput
- o MC_ECATIOReadDigitalInput
- o MC_GetReactorStatistics

DONE bit was removed, and instead a valid bit was added to the end of the Outputs list, in the following Function Blocks / Functions:

- o MC_GlobalReadBoolParameter
- o MC_GlobalReadParameter

New Valid bit added to the following function block:
- o GetOpMode

New Function Blocks

- o MC_PathSelect
- o MC_MovePath

ElmoTypes Modifications and additions:

NC_ARC_SHORT_LONG_ENUM:	Add MC_NONE_ARC_CHOICE
NC_AXIS_IN_GROUP_TYPE_ENUM:	Major modification
NC_PATH_CHOICE_ENUM:	Add MC_NONE_PATH_CHOICE
NC_TRANSITION_MODE_ENUM:	Added:

MC_TM_CORNER_DIST_TC_POLYNOM
MC_TM_CORNER_DIST_CV_POLYNOM3
MC_TM_CORNER_DIST_CV_POLYNOM5

New Enable bit instead of Execute bit were added to the following function blocks:

- o ElmoGetFloatArr
- o ElmoGetFloatParam
- o ElmoGetIntArr
- o ElmoGetIntParam:

In the MC_ReadStatus a new Pending bit was added to the end of output list

The following API's were removed:

- o MC_ReadDigital32BitOutputs
- o MC_ReadDigitalOutputs

PLC Open adjustments

Small adjustments were made to the inner state machine of the function blocks, in order to be 100% compatible with PLCpen standard.

> *With Execute***:** The "Done", "Error", "ErrorID" and "CommandAborted" outputs are reset with the falling edge of "Execute". However the falling edge of "Execute" does not stop or even influence the execution of the actual FB. It must be guaranteed that the corresponding outputs are set for at least one cycle if the situation occurs, even if execute was reset before the FB completed.
> If an instance of a FB receives a new execute before it finished (as a series of commands on the same instance), the FB won't return any feedback, like 'Done' or 'CommandAborted', for the previous action.

> *With Enable***:** The "Valid", "Enabled", "Busy", Error",and "ErrorID" outputs are reset with the falling edge of "Enable" as soon as possible.

1.6 GMAS Home on Block Support

This new additional Homing method allows users executing a built-in Homing method on mechanical Hard Stop, when no limits or index is available. The method is defined by the PLCOpen Standard. Users can define the homing speeds, Max Torque limitation for the hard-stop, travel limit, and timeout for the process.

The new function is also supported by the underplaying DS-402 at drive level as well.
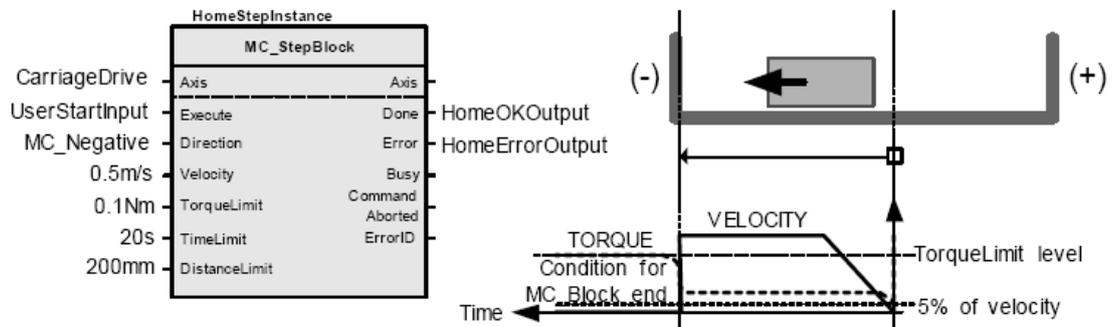
- o The standard GMAS API HomeDS402 is used.
- o The homing methods to be sent are as follows:
  - Reverse Home On Block ➔ -1
  - Forward Home On Block ➔ -2
- o The torque limit should be always positive.
- o The distance limit should be always positive.

Release Notes - New Firmware Version for GMAS

Version 1.1.0.6

And Library Update 226

1. General

The "***uImage_v1.1.0.5_2012_03_14.gms***" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. Addition of the following API's to 'C' and CPP libraries (and of course support in GMAS):
   MMC_GetErrorDescriptionByID - This function receives an error / warning code and returns the description and resolution for this error/warning.(taken from the Personality file)
   MMC_GetGroupMembersInfo - The function returns information regarding the members of a desired group:
   - o Number of axes in group (members).
   - o Axis names.
   - o Axis references.
   - o Device ID's.
2. **CPP Error Handling** – New functionality was added to the error handling in CPP. When we register the RTE function callback we should define whether we want to close the program in case of error or not.
3. New functionalities were added to the "**SetKinTransform**" function:
   - a. **New error (-203)**.

     We currently do not support the following axis direction types:

     Polar type (U,V,W).

     S type.

     In case the user tried to configure an axis to one of the unsupported types an error will

     be generated (NC_KINEMATIC_DIRECTION_NOT_SUPPORTED (-203)).

   - b. **New error (-204)**

     When you work with the 'N' axes type, the user must make sure that they were defined consecutively.   If the user chooses to use 3 N axes, the user should choose the N1,N2,N3 axes.

     The order of the Kinematic definition does not have to be in the order of "N" axes definition.

So you can define the Kinematic as N3,N2,N1 (as long as there are no 'holes'.

Example:

    Good: (N1,N2,N3), (N1,N2,N1), (N1,N3,N2)

    Bad: (N1,N5,N6), (N2,N3,N4), (N3,N2,N3)

If the N axes were not defined consecutively, an error will be generated (NC_N_AXES_ARE_NOT_CONSECUTIVE (-204)).

c.   **New error (-207)**

Currently, we do not support the case when the number of axes that participate in the kinematic group is smaller than number of axes in the group.

If the user tries to define a Kinematic (using SetKinTransform function)

where the number of axes in the kinematic group is smaller than number of the

members in the group, the following error will be generated (NC_ALL_AXES_SHOULD_BE_IN_KINEMATIC (-207)).

d.   **New error (-208)**

The GMAS does not support a definition of only one Cartesian direction.

The GMAS does not support a Cartesian coordinate system with only one direction(X,Y or Z).

If the user tries to define a Kinematic with Only one Cartesian direction, the following error will be generated: (NC_NOT_SUPPORT_ONE_CARTESIAN_AXIS (-208)).

In this case you can simply run the linear FB's in ACS mode.

The disadvantage of using ACS is that you cannot use the transformation mechanism.

Another solution for this case is to use this axis as an 'N' axis.

e. **New error (-209)**

Currently, we support only the following coordinate systems:

* ACS.

* MCS.

If the user tries to any other coordinated system (PCS), the following error will be generated (NC_NOT_SUPPORTED_COORDINATE_SYSTEM(-209)).

f. **New error (-210)**

No Blending Support in ACS mode. When inserting a Function Block in ACS mode no blendings can be set.

Please set all ACS FB's:

* In buffered or abortion mode.

* Without any Transition mode, with the exception of MC_TM_NONE_MODE.

If the user tries to set any blending mode in ACS mode, the following error will be generated: NC_NOT_SUPPORT_BLEND_TRANS_ON_ACS(-210)).

g. **SetKinTransform Improvement**.

This improvement is related to zeroing of parameters in the beginning of the

function, to avoid some trash from the "Previous" SetKinTransform call.

This change is very important when you define two different kinematics

to the same group with a smaller number of members.

Example:

 * Define group with 5 axes.

 * Set Kinematic (for 5 axes).

 * Remove one axis from group (dynamically, using MMC_RemoveAxisFromGroup function).

 * Set Kinematic (for 4 axes).

h. **Update of the NC_AXIS_IN_GROUP_TYPE_ENUM enum**.

Add new enumerators

```
typedef enum

{

        NC_X_AXIS_TYPE = 0,

        NC_Y_AXIS_TYPE,

        NC_Z_AXIS_TYPE,

        NC_U_AXIS_TYPE, - NEW

        NC_V_AXIS_TYPE, - NEW

        NC_W_AXIS_TYPE, - NEW

        NC_N1_AXIS_TYPE,

        NC_N2_AXIS_TYPE,

        NC_N3_AXIS_TYPE,

        NC_N4_AXIS_TYPE,

        NC_N5_AXIS_TYPE,

        NC_N6_AXIS_TYPE,

        NC_N7_AXIS_TYPE,

        NC_N8_AXIS_TYPE,

        NC_N9_AXIS_TYPE, - NEW

        NC_S_AXIS_TYPE,  - NEW

        NC_LAST_AXIS_TYPE

}NC_AXIS_IN_GROUP_TYPE_ENUM;
```

4. Bug Fix. When working in mult- axis with transition mode 7 or 8 and your system has one of the
    following axes, N1-N9 and S – Unexpected profile creation was detected. This was fixed.
5. Profiler Changes:
    a. 'N' axes support in MCS.
    b. Code optimization and improvement

      c.   Stop using transition mode 6 (even in 3D case)

6. **Migration to Revision 2.1 of uProcessor from Revision 1.0**. this version supports both revisions. For further information, please contact Elmo.

Release Notes - New Firmware Version for GMAS

Version 1.1.0.5

2. General

The "**uImage_v1.1.0.5_2012_03_14.gms**" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

7. Stopping the ongoing Spline FB DC and Jerk is now taken from the *Stop* function block and not the default DC and Jerk parameters.

8. Profiler fixes. In some profiles that, in the end of a FB that is blended by another FB, the AC / DC is not continuous and had small jumps. This was fixed.

9. The Circle Angle mode can be operated only in 2D case. The verification of the number of Cartesian axes was fixed and improved.
    For example, 6 axes in a group, 1 as "X" 2 as "Z" 1 as N1 and 2 as N4. (This is an allowed case).

10. In case when we try to Enable the Group, all the axes should be in StandStill state. If one of them is not in Stand Still state we will return the following error:
    NC_ONE_GRP_MEMBER_IS_DISABLED and not the former - NC_UNSUITABLE_NODE_STATE error.

11. SetPosition function will return the following error: NC_MODE_NOT_SUPPORTED.

12. In function "Remove Axis From Group" when you try to remove an axis that was not included
    in the group (a particular case is when you try to remove axis from an empty group) the following error will be returned: NC_NODE_NOT_FOUND (instead of NC_MAX_MEMBERS_IN_VECTOR).

13. Profiler bug fix in Single Axis case. When you work in Single Axis with blended buffer mode, sometimes the profiles were built wrongly. We get position and / or velocity overshoots. This was fixed.

14. New error code - NC_WRONG_EVENT_MODE   (-187)

    This error is generated from the following functions:

    ConfigPDO3EventMode
    ConfigPDO4EventMode

    Meaning:
     An incorrect event mode input was sent to the function.

15. New error code - NC_SUPPORTED_ONLY_ON_3D_CASE (-188)

    This error is generated in case when there was an attempt to run a function that can be

executed only in 3D case but the system is not in 3D configuration.

For instance: MoveCircularAbsoluteCmn in Radius mode.

16. Bug Fix - Wrong axis reference was sent on the Emergency callback, on EtherCAT.

17. EoE timeouts when downloading files to slaves bug fix.

18. Automated Distributed Clock (DC) calculations. DC is now automatic and all cycle times, from 500us until 10ms (Intervals of 250us) were tested and work properly.

19. USB related problems

    a. Now there is no printout when disconnecting and connecting gadget serial. This caused latencies on other processes / threads in the system.

    b. When disconnecting the gadget serial, the child processes are no longer killed. Only the gadget serial udev process is killed.

Release Notes - New Firmware Version for GMAS

Version 1.1.0.4

And Library Update 225

3. General

The "***uImage_v1.1.0.4_2012_02_15.gms***" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. New axis (Single and Group) status Pending bit.
2. Updated Personality.
3. Ability to download files to GMAS via tftp to usr directory.
4. New behavior on FB insertion mechanism
5. New API – GetActiveVectorsNum
6. New initial maximum jerk values.
7. SetKinTransform function protections.
8. Working with coefficients bug fix.
9. PathChoice data verification in MoveCircular functions
10. SendSDO ulDataLength parameter validation.
11. Splines – ACS support.
12. TargetPosition and DesiredPosition are now synchronized.
13. New method of releasing FB's in GMAS and related APIs
14. Garbage Collection mechanism when working via IEC.
15. Additional VelocityOffset parameter can be mapped to work with new drive versions via CyclicPosition operational mode.
16. Even more stabilized profiler.
17. New error codes related to MCS
18. Bug fixes related to end motion with non empty function block queues.
19. PDO mapping modifications sequence in CAN.

3.1   New axis (Single and Group) status Pending bit

This bit indicates that the last Profiler related FB was entered with execute bit "0" (relevant only in NC and Virtual mode). The bit mask is :

#define NC_AXIS_PENDING_MASK   0x10000000   ///< Axis/Group Pending Bit Mask

It appears in MMC_definitions.h header file.

3.2   Updated GMAS Personality

- AC\DC record parameter was removed from Group
- AC\DC record parameter was removed from Axis

3.3   Ability to download files (tftp) to GMAS to usr directory

An additional ENUM was added to the MMC_DOWNLOAD_TYPE_ENUM types:

typedef enum

{

    MMC_PARAMETER_FILE_DOWNLOAD = 0,   ///< Down load parameter file.

    MMC_RESOURCE_FILE_DOWNLOAD,   ///< Down load resource file.

    MMC_SNAPSHOT_RESOURCE_FILE_DOWNLOAD,   ///< Down load resource snapshot file

    MMC_ETHERCAT_CFG_FILE_DOWNLOAD,   ///< Down load resource ethercat file

    MMC_PERSONALITY_FILE_DOWNLOAD,   ///< Down load G-MAS personality file

    MMC_USER_FILE_DOWNLOAD,   ///< Down load G-MAS user file

}MMC_DOWNLOAD_TYPE_ENUM;

This enables the user, to download files, via the MMC_ResImportFileCmd API function, directly to the /mnt/jffs/usr folder. This is useful for data files, spline data, etc ... and is to be incorporated within the EAS.

3.4   New behavior on FB insertion mechanism

- New behavior for Pending Blended Function block (Multiaxis only) - When trying to insert a FB with execute bit "0" after a FB with execute bit "1" (we call this situation - *pending*), the GMAS enables the user to insert the second FB in blended mode with a transition only in case the global parameter "MMC_SUPPORT_BLENDED_FB_PENDING" (53)equals to "1" (true) otherwise we will insert the second FB in "MC_BUFFERED_MODE" mode.

  The global parameter "MMC_SUPPORT_BLENDED_FB_PENDING" is initialized to "0" and can be changed using one of the following API functions:
  - a. MMC_WriteParameter.
  - b. MMC_GroupWriteParameter.

- New behavior when Inserting a Blending FB to active FB – It is not possible to insert a FB with blended mode (with transition arc) when the last FB in queue is active (G-MAS executes the last FB). In this case we will change the BufferMode to "MC_BUFFERED_MODE" and the TransitionMode to "MC_TM_NONE_MODE" and not insert the arc but only the original FB. If the user wants to persist that the FB is inserted with an arc and avoid this situation, insert the previous FB with execute bit "0". An additional way to know the status of the FB list is to use "MMC_GetFBDepth" API function. If you receive a number greater than one it means that the last FB is not during execution.

3.5   New API – GetActiveVectorsNum

This function returns the actual number of Vectors (Groups) in the GMAS

3.6   New Initial Maximum jerk values

The initial value of maximum jerk was modified from 1E12 to 1E15

3.7   SetKinTransform Function Protections

- SetKinTransform function can be called in the following states only:
  - a. GROUP_STANDBY.
  - b. GROUP_DISABLED.

  An appropriate error is returned (-182).

3.8 Working with coefficients bug fix

A bug was fixed related to the work with coefficients, correct updating of positions after PowerOFF\ON and after GroupDisable\Enable. Also the GMAS now supports the changing of coefficients without restarting the axes and group.

3.9 PathChoice Data verification in MoveCircular functions

Path choice and arc short long parameters data verifications were added in the MoveCircular functions.

The verification logic is as follows:

Check the validity of the path choice and arc short long parameter according to the circle mode

The following parameters are relevant (in each mode):

MC_BORDER_CIRC_MODE   -   None

MC_CENTER_CIRC_MODE   -   eArcShortLong

MC_RADIUS_CIRC_MODE   -   eArcShortLong and ePathChoice

MC_ANGLE_CIRC_MODE   -   None

3.10 SendSDO ulDataLength parameter validation

The data length parameter the SendSDO function receives - is and always was in bytes and NOT bits.

A bug was reported by SQA– and as a result we found that when the user sends a request to upload/download an SDO object that is **not** 1,2 or 4 bytes – **4 bytes are always sent.**

This bug was fixed in the following way: If the user requested to up/down an SDO object that is **not** 1,2, or4 bytes – an error will be returned (NC_WRONG_FUNC_ARGUMENT_TYPE).

It is important to note, as stated above: If the user previously wrote code that requested a SendSdo of inappropriate data size, the function would have sent/requested 4 bytes. Now any error is returned.

3.11 Splines – ACS support

Splines now support ACS mode as well as MCS. This means that more than 3 axes can participate in a spline motion.

3.12 TargetPosition and DesiredPosition are now synchronized

The data recording parameters:

- TargetPosition (Communication parameter).
- DesiredPosition (Profiler output).

Had a difference in value of 1 real time cycle. This was fixed in this version.

3.13 New method of releasing FB's in GMAS and related APIs

The method of how the GMAS releases the function blocks to the *free* state, depended on a specific setting in the resource file.
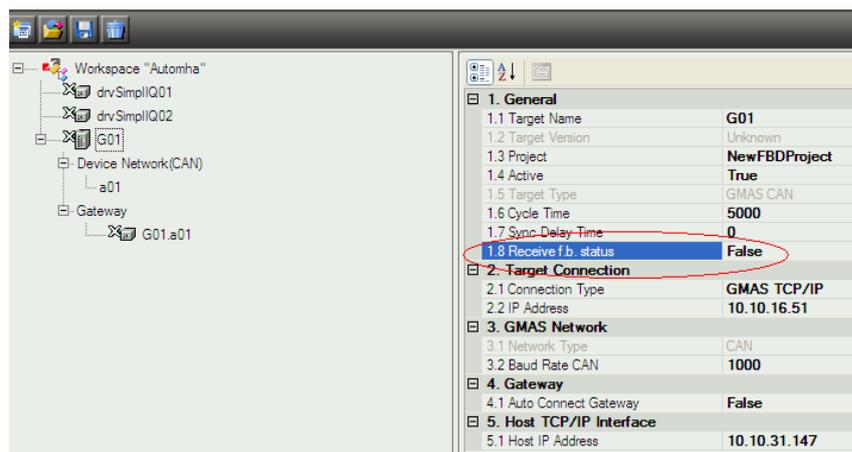
This mechanism caused the following problems:

- User never knew how to work. When to set this flag.
- There were times when the flag was set, and the user could not perform any motions due to *no free* function blocks.

In order to solve these problems, the following was implemented:

- The Flag, as appears above, only specifies the default value for the User program running in IPC.
- All RPC FB's inserted, are released as soon as the motion received the *done* flag.
- The user no longer needs to set this flag when working in IEC. It is automatically set, via the *MMC_SetEnquireFbStatusCmd* and *MMC_GetEnquireFbStatusCmd* API's in the IEC runtime.

3.14 Garbage Collection mechanism when working via IEC

The IEC runtime performs an advanced algorithm for releasing function blocks. This is essential for function blocks the user did not perform any "Inquiries" on.

3.15 Additional VelocityOffset parameter can be mapped to work with new drive versions via CyclicPosition operational mode

As per the CyclicPosition DS402:
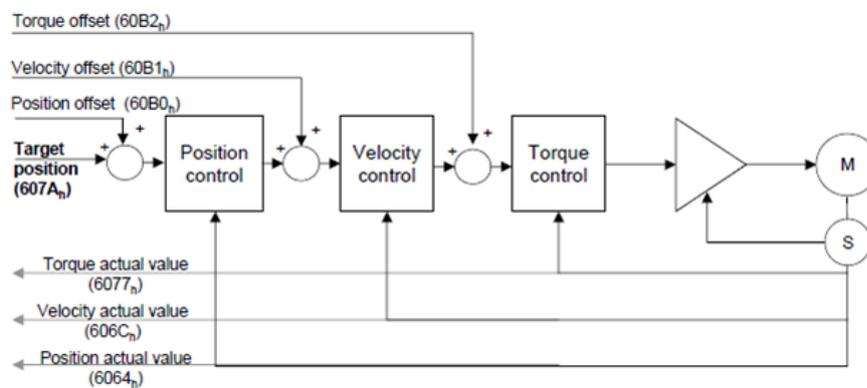
The velocity offset 0x60b1 can be added (Gold Ethercat only) in order to send a PVT command (instead of the PT that is sent today). This drastically improves the motion performance.

In order to use this parameter, it must be mapped via the Ethercat configuration tool in the EAS:
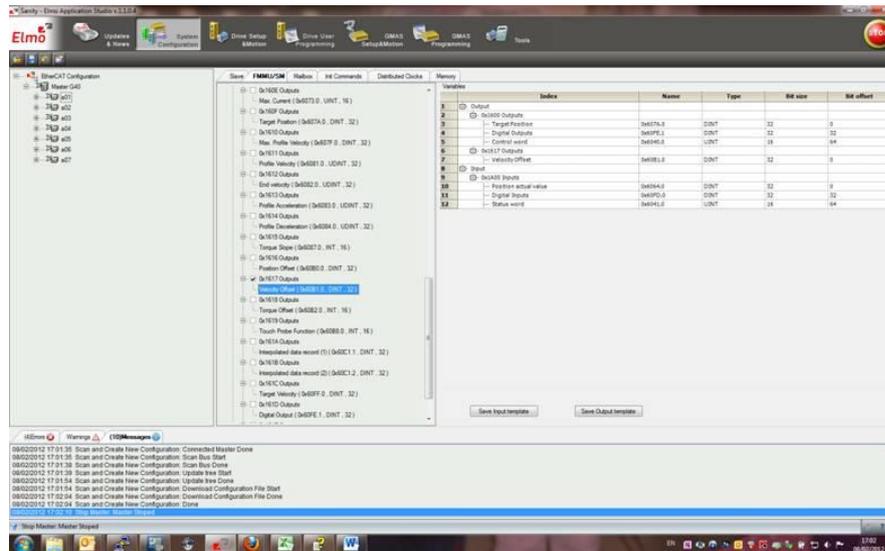
3.16   Even more stabilized profiler

Bug fixes related to the GMAS profiler were fixed. These bugs were related to overshoots and Speed overrides that were used.

3.17   New error code related to MCS

When you configure your MCS kinematic system (in the SetKinTransform function) you should provide the Forward and Backward coefficients.

In this transformation function, the ratio between the coefficients should be:

NC_BACK_TR_RATIO_COEFxNC_BACK_SHIFT_COEF = 1

The maximum permitted deviation is 1E-5, otherwise an error is generated (-181).

3.18 Bug fixes related to end motion with non empty function block queues

Added support to cover cases when motion is finished with NON-Empty FB queue, but all the remained FB in the queue are with execute bit "0".

3.19 PDO mapping modifications sequence in CAN

Valid bit logic was added to the PDO mapping sequence.

The valid bit was not used (and is actually ignored) in the SimplIQ. Additional logic was added to the GMAS in order to support the CAN mapping in the Gold drives.

However – the CancelConfigPDO function must be called prior to calling CongiPDO3-4.

Release Notes - New Firmware Version for GMAS

Version 1.1.0.1

4. General

The "***uImage_v1.1.0.1_2012_01_05.gms***" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. Additional Multi axis transition mode support.
2. Splines support.
3. Error correction support via CAN.
4. Remote IEC start /stop API's.
5. Save / Load parameters support
6. FoE support,
7. Bug regarding networking mask.
8. Digital IO data recording support
9. Abortion buffer mode bug fixes.

4.1 Additional Multi axis transition mode support

Two additional transition modes were added in this version:

- MC_TM_CORNER_DIST_CV_POLYNOM3
- MC_TM_CORNER_DIST_CV_POLYNOM5

These modes were added to the already existing transition mode enumerations.

Transition mode MC_TM_CORNER_DIST_CV_POLYNOM3 = 7

Note: **The Transition parameter must be defined as a corner distance.**

In this mode, the transition curve is constructed as a cubic polynomial of some parameter "s". The initial estimation of the parameter variation is based on the distance between start and end-points, with the aim to approach the desired curve length.

The transition is executed with a constant velocity. This velocity originally defined by the blend mode, may be decreased if due to the trajectory parameters, it cannot be achieved. It can also be decreased before the transition curve is built, if the predefined velocity at some points of the transition curve, causes acceleration greater than the one defined by the parameter MMC_MAX_ACCELERATION_PARAM. This acceleration is calculated as a = v^2/ρ where ρ is a radius of curvature.

This transition mode guarantees transition smoothed by the velocity, but produces a minor jump via acceleration at start and end-points of the transition curve. If the acceleration achieved at some points of the

transition curve is less than one predefined by the parameter MMC_MAX_ACCELERATION_PARAM but greater than desired, the user can decrease the value of the parameter causing a decrease of the transition velocity by the formula

$V = [\text{MMC\_MAX\_ACCELERATION\_PARAM} *\rho_{min}]^{1/2}$

where ρ_{min} is the minimal radius of curvature on the transition curve.

This mode is recommended in some issues with the mode 8, e.g. detrimental geometry.

Transition mode MC_TM_CORNER_DIST_CV_POLYNOM5 = 8

Note: **The Transition parameter must be defined as a corner distance.**

In this mode the transition curve is constructed as a quintic polynomial of some parameter "s". The initial estimation of the parameter variation is based on the distance between start and end-points with the aim to approach the desired curve length.

The transition is executed with a constant velocity. This velocity originally defined by the blend mode, can be decreased if due to the trajectory parameters, it cannot be achieved. It may also be decreased before the transition curve is built, if the predefined velocity at some points of the transition curve, causes acceleration greater than the one defined by the parameter MMC_MAX_ACCELERATION_PARAM. This acceleration is calculated as $a = v^2/\rho$ where ρ is a radius of curvature.

This transition mode guarantees transition smoothed by the velocity and acceleration. If the acceleration achieved at some points of the transition curve is less than one predefined by the parameter MMC_MAX_ACCELERATION_PARAM but greater than desired, the user can decrease the value of the max acceleration parameter causing a decrease of the transition velocity by the formula

$V = [\text{MMC\_MAX\_ACCELERATION\_PARAM} *\rho_{min}]^{1/2}$

where ρ_{min} is the minimal radius of curvature on the transition curve.

This mode is recommended for 3D transitions, and for 2D transitions if the inevitable acceleration jumps for the transition by the circle arc are unacceptable

For instance a simple 2D example:

The example contains two lines with 90 degrees between them:
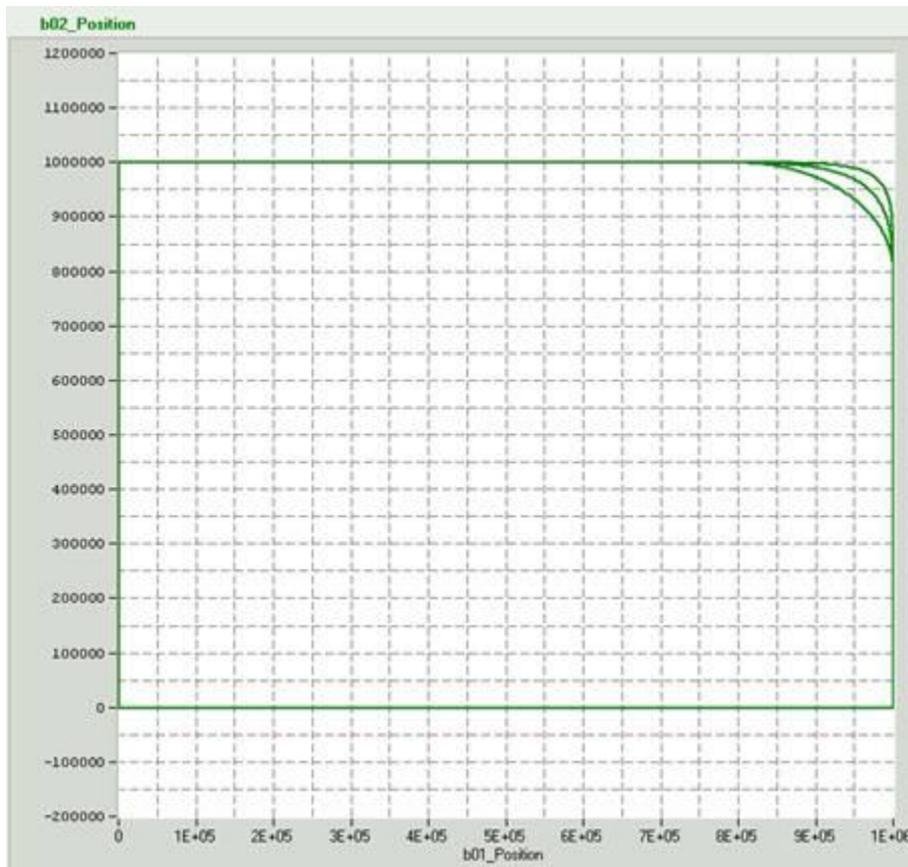
The sequence is:

- Start from 0,0
- Go to 0,1000000
- Go to 1000000, 1000000
- Go to 1000000,0 (**Blended**)
- Go to 0,0

The sequence was performed three times.

- First time with Transition mode "3".
- Second time with Transition mode "7".
- Third time with Transition mode "8".

If we just see at the position plots we cannot indicate any significant difference between them.

We only can say that in higher transition mode we get closely to the desired point (1000000,1000000).
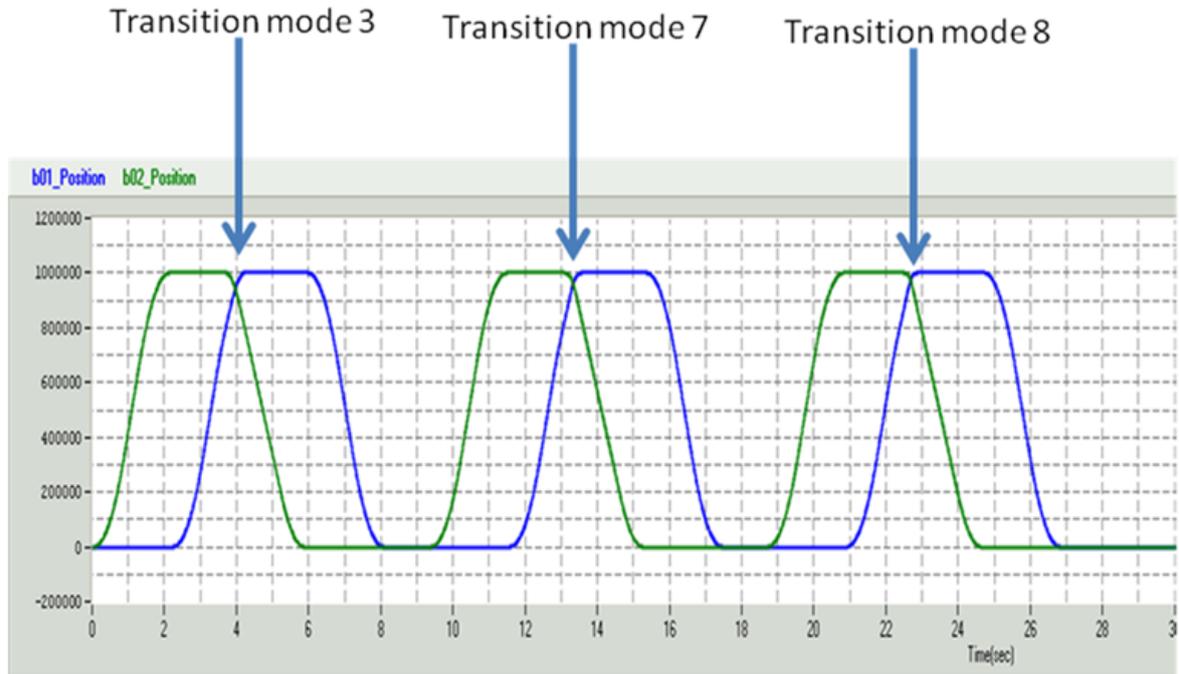
To see and understand the real difference between transition modes 3,7 and 8 we should look at the Velocity and AC/DC graphs.

The keyword of the differences is "CONTINUITY", as the transition mode is higher – the continuity is "better" (higher order).

- In transition mode 3 – the continuity is only in position. You can see than in all cases you have continuity in position, there are no unexpected jumps or sharp transitions.
- In transition mode 7 – the continuity is in position and velocity. You will see below than on this mode the continuity is in position and velocity.
- In transition mode 8 – the continuity is in position velocity and AC/DC. You will see below than on this mode the continuity is in position velocity and AC/DC.

Let's look at the graphs of each transition mode (Position, Velocity and AC\DC)
The red circles indicate the points of NON-continuity.

Transition Mode 3
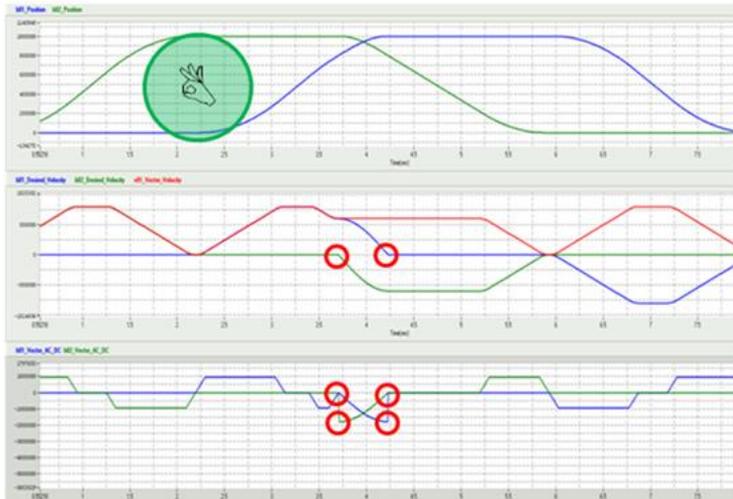
Transition Mode 7
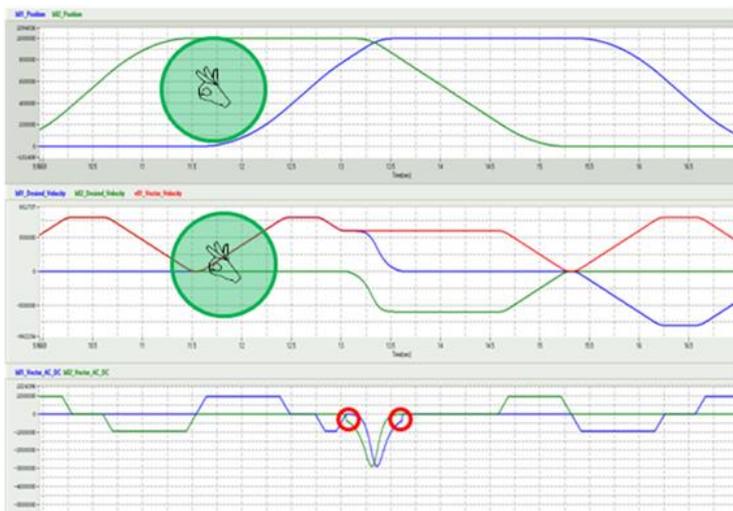
Transition Mode 8
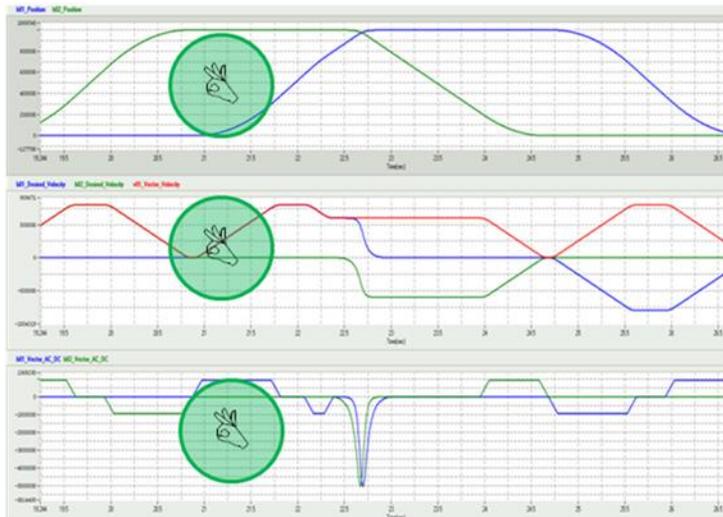
4.2   Splines support

The functions associated with Spline; MMC_Movepath, MMC_PathSelect, and MMC_Unselect use quite specific user defined parameter modes to describe the transition. These are:

MC_SPLINE_MODE_FT = 1

MC_SPLINE_MODE_V_AC_DC = 2

MC_SPLINE_MODE_NP = 3
MC_SPLINE_MODE_CV = 7

These API's were added to the MMC_API and are now standard features for the GMAS. Please refer to the user manual for further reference.

4.3   Error correction support via CAN

Error correction mechanism for CAN motion devices is now part of the GMAS firmware.

4.4   Remote IEC start /stop API's

API's were added in order to support she following IEC runtime operations via the MMC_API:

- Start IEC runtime.
- Stop IEC runtime.
- Is runtime running.
- Start IEC runtime at startup.
- Stop IEC runtime from Startup.

These APIs are not exposed to the user,

4.5 Save / Load parameters support

There is now full support GMAS parameters:

- Loading GMAS parameters at powerup. If a file was previously saved it will automatically be loaded. Default values are set if they are not in a permitted range.
- Loading GMAS parameters at request - MMC_LoadParamCmd will take the parameters from the GMAS FLASH and load it into the GMAS memory.
- Saving GMAS parameters - MMC_SaveParamCmd will save the current parameters to the GMAS FLASH.
- Uploading and downloading the files to the GMAS is now available via the Import/Export resource file interfaces.

4.6 FoE support

FULL FoE support was added in this GMAS version. in order to support this, the following new API's were added:

- MMC_DownloadFoE. For further details please contact Elmo.
- MMC_GetFoEStatus. For further details please contact Elmo.

These functions can be called ONLY after the network was properly configured via the Ethercat configuration tool.

1.1   Enhanced Ethercat Statistics Function

This new GMAS version includes a new API function MMC_GetEthercatCommStatistics returning data for required Ethercat nodes:

- o Send / Receive / Parse errors.
- o Wrong working counters.
- o Number of slaves identified on network.
- o Master states and diagnostics.
- o For all required slaves:
    - Slave state and diagnostics.
    - Vendor, Product and Rev numbers.

1.2   Bug regarding networking mask

The GMAS now supports class 'B' network types.

1.3   Digital IO recording support

There is now support to record Digital IOs from data recording interface. This is also updated from within the personality

1.4   Abortion bug fixes

There were many cases where the abortion buffer type did not work, and this is a major regression fix. The following phenomena's were noticed:

- Abortion as part of first function block did not work.
- Aborting an ongoing motion had unexpected behavior.
- Aborting on negative positions did not work.

This was all fixed as part of the version release.

Release Notes - New Firmware Version for GMAS

Version 1.0.3.3 and MMC_API_LIB version 219

2. General

The "*uImage_v1.0.3.3_2011_11_14.gms*" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. Profiler fix – generate warning in case of zero length (Single Axis).
2. Ended motion event. We now support end of motion event on Group as well. The mechanism was modified/rewritten in order to support both cases in an optimized manner.
    a. Single Axis – same interface for the user.
    b. Group – add support for group end motion event.

    c. In order to enable/disable the event the same function as in single axis is to be used:

3. MMC_EnableMotionEndedEventCmd in 'C'.

4. Node Error event – A new event was added for axis entering the error state.
    a. There are new event enumerator in the "C" API.

    b. Added support for this event to the "CPP" wrapper as well.
    c. To configure event handler function use "RegisterEventCallback" function in Connection
    d. class.
    e. The relevant enumerator for registration is "MMCPP_NODE_ERROR".
5. Bulk Read Mechanism – Mechanism for reading bulk data from the GMAS. Available from host communication and internal – IPC communication. Increases throughput and utilizes bandwith.
6. MoveLinearAdditiveCmd API. The MoveLinearAdditiveCmdAPI is not standard PLCopen and is now available via the GMAS API.

2.1 Profiler fix – Generate warning in case of zero length

A warning is now generated to the user in case the Single Axis motion was not performed.

In this case - the warning 1009 is returned.

2.2 Group Ended motion event

A new mechanism was built in order to support group motion ended in addition to the single motion ended event. Initializing the event remains the same as before:

MMC_EnableMotionEndedEventCmd in 'C'

From the C++ interface, new APIs were added in order to support this in the CMMCAxis class:

- CMMCAxis::EnableMotionEndedEvent
- CMMCAxis::DisableMotionEndedEvent

The same callbacks are called as in the Singe axis – but now the axis reference of the group is called.

2.3 Node Error event

A new event was added for axis entering the error state, in addition to the many other event types:

#define ASYNC_REPLY_EVT   0

#define DOWNLOAD_FW_EVT 4

#define EMCY_EVT   5

#define MOTIONENDED_EVT   6

#define HBEAT_EVT   7

#define PDORCV_EVT   8

#define DRVERROR_EVT   9

#define EMIT_EVT   10

#define HOME_ENDED_EVT   11

#define SYSTEMERROR_EVT   12

#define MODBUS_WRITE_EVT 13

#define TOUCH_PROBE_ENDED_EVT   14

#define NODE_ERROR_EVT   15

In the C++ interface, a new callback for entering the Error State was added. This needs to be initialized similar to any other callback initialization. The RegisterEventCallback function now receives a new value - " MMCPP_NODE_ERROR" in the eClbType parameter stating the users callback function. The callback function returns the last emergency code and the error code regarding the reason for entering the error state.

Please note: If the emergency callback is also initialized, the user may experience calls to both the *error* and *emergency* event callbacks.

2.4 Bulk Read Mechanism

Until now, all needed parameters can be retrieved by API calls. The user has to implement the parameters retrieval in his code (for example read: current, position, etc.). If the user wants to get a "bulk" of parameters – it needs to be retrieved one by one. This operation is relatively slow, especially from a remote host, as a single TCP/IP packet is used in order to transfer at most a "double" variable. Via the user program, the operation is also wasteful, for the same reason.

In order to "speed things up" – a *Bulk Read* Mechanism was built. This mechanism allows the user to choose from a set of *preset* parameters, and choose which axes to read from. In addition – the user may create his own *data* to read by manually configuring the data.

The following 'C' APIs were added:

- MMC_ConfigBulkReadCmd - Configures the function to read all paramaters from multiple axes
- MMC_PerformBulkReadCmd - Reads all paramaters from multiple axes

The following C++ API's were added:

- CMMCPPGlobal::ConfigBulkRead
- CMMCPPGlobal::PerformBulkRead

2.5 Group Move Additive API

In addition to the MoveAdditive API supported in SingleAxis (actually part of the PLCopen standard) – the GMAS now supports MoveLinearAdditiveCmd API.

This group motion API, similar to the single motion API - Commands a controlled motion of a specified relative distance **additional** to the most recent **commanded** position in the discrete motion state.

This is opposed to the **MoveRelative** function block which commands a discreet controlled motion of a specified distance **relative** to the set position **at the time of the execution**.

The following 'C' APIs were added:

- MMC_MoveLinearAdditiveCmd

The following C++ API's were added:

- MoveLinearAdditive

Release Notes - New Firmware Version for GMAS

Version 1.0.3.2

3. General

The "*uImage_v1.0.3.2_2011_10_11.gms*" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. Setdefaultparameters API.
2. SetKinTransform - Bug Fix.
3. Corner distance Polynomial transition mode.

The Following 'C' APIs were added to this version:

- MMC_SetDefaultParametersAxisCmd
- MMC_SetDefaultParametersGlobalCmd

The Following C++ APIs were added to this version:

- SetDefaultManufacturerParameters (Equivalent to MMC_SetDefaultParametersAxisCmd 'C' function)

3.1 Reverting GMAS Parameters to Factory Defaults

The following APIs were added:

- MMC_SetDefaultParametersAxisCmd - This function set default manufacturer parameters to the Axis related parameters.
- MMC_SetDefaultParametersGlobalCmd - This function set default manufacturer parameters to the Global / Axis related parameters.

3.2 MMC_SetKinTransform Bug Fix

A bug was detected and fixed in the MMC_SetKinTransform function. The function did not properly detect the members of the group and in some cases returned errors. This bug was fixed.

3.3 Corner Distance Polynomial

A new transition type was added to the list of transition types:

typedef enum

{

MC_TM_NONE_MODE	= 0,
MC_TM_MAX_VELOCITY_MODE	= 1,
MC_TM_DEFINED_VELOCITY_MODE	= 2,
MC_TM_CORNER_DISTANCE_MODE	= 3,
MC_TM_MAX_CORNER_DEVIATION_MODE	= 4,
MC_TM_SWITCH_RADIUS_MODE	= 5,
MC_TM_CORNER_DIST_TC_POLYNOM	= 6,

}NC_TRANSITION_MODE_ENUM;

The MC_TM_CORNER_DIST_TC_POLYNOM transition calculates a 5th order polynomial calculation on the corner distance. But beware: The velocity is reduced as a result of the calculation and is inherent in this mode. A new transition mode is to be introduced in the near future that does not reduce the velocity.

Release Notes - New Firmware Version for GMAS

Version 1.0.3.1

4. General

The "***uImage_v1.0.3.1_2011_09_26.gms***" is a new firmware release for the GMAS Master Motion Controller, which supports the following new major features:

1. GMAS Personality
   a) Data recording parameters.
   b) Error list and resolutions
   c) GMAS parameters description.
2. Save / Load Parameters per axis.
3. PDO Information API. Ability to read what is currently configured.
4. Sync Delay time as EAS parameter
5. New Modbus warnings
6. Specific error to Out of range parameters
7. Warning if in current power state, current operation mode
8. Zero Len profile handling warnings

The Following 'C' APIs were added to this version:

- MMC_GetPDOInfoCmd

The Following C++ APIs were added to this version:

- GetPDOInfo 'C' function) – calls the MMC_GetPDOInfoCmd 'C' function.

4.1 GMAS Personality

A new XML formatted file, by the name of /var/MMC/config/resources/Personlity.xml is introduces in this version. this file includes information regarding the following:

- Data recording parameters – From time to time, the GMAS updates its data recording parameters. Instead of performing modifications in the host software (i.e EAS) in order to support these changes, the list of parameters, including the: size, units, and type appear in this file.
- Error List and resolution – When an error is returned from the GMAS, either to a host or user program – only an error ID is returned. The file includes a full description regarding the error and a resolution to the error. Via the CPP interface, the user automatically receives a printout including a full description from the XML file with the description of the error and the resolution.
- GMAS parameters description – There is a list of GMAS parameters which is to be viewed in the EAS. Each parameter has the following attributes:
    - o Number.
    - o Name
    - o Type.
    - o Array size.
    - o Permissions (RO, save to flash, etc …).
    - o Min / MAX / Default values.

In order to upload the personality, an additional enum was added as per the personality file:

enum

{

      MMC_PARAMETER_FILE_DOWNLOAD = 0,

      MMC_RESOURCE_FILE_DOWNLOAD,

      MMC_SNAPSHOT_RESOURCE_FILE_DOWNLOAD,

      MMC_ETHERCAT_CFG_FILE_DOWNLOAD,

      MMC_PERSONALITY_FILE_DOWNLOAD

};

4.2 Save / Load Parameters per axis

After changing parameter values via the SetParameter APIs (or via the designated EAS window), the parameters, per axis can be saved to the GMAS FLASH. These parameters are saved to: /mnt/jffs/MMC/config/parameters folder on the GMAS. The name of the file is the name of the axis in the resource file with the "prm" extension.

The logic of loading the parameters is as follows:

- Load parameters default values.
- If the parameter file exists for a specific axis, load the file and set all Non-Read only parameters .

4.3 PDO Information API

In order to read which PDOs are configured via the CAN network, a new API was created:

MMC_GetPDOInfoCmd

This API returns data (type, SYNC, TPDO and RPDO data) for a designated PDO (3 or 4).

4.4 Sync Delay time as EAS parameter

A new parameter was added to the resource file, and can be accessed from the main page in the EAS (GMAS activity) by the name of *Sync Delay Time*. This parameter defines the delay between the SYNC and the *Target Position* data when working in DS402 Interpolated Position operational mode.

4.5 Specific Error to Out of Range Parameters

Until this version release, a general *Out Of Range* error was returned when a motion command was sent (NC_PARAM_OUT_OF_RANGE   (-22)). The following list states specific errors per parameter that is out of range:

#define NC_SET_IS_TO_LOAD_PARAM_OUT_OF_RANGE   (-136)

#define NC_SET_OVERRIDE_OUT_OF_RANGE   (-137)

#define NC_NUM_OF_AXES_OUT_OF_RANGE   (-138)

#define NC_TR_FUNC_TYPE_OUT_OF_RANGE   (-139)

#define NC_BUFFERED_MODE_OUT_OF_RANGE   (-140)

#define NC_HOMEDS402_METHOD_OUT_OF_RANGE   (-141)

#define NC_DIGITAL_INPUT_NUMBER_OUT_OF_RANGE   (-142)

#define NC_MAX_VELOCITY_OUT_OF_RANGE   (-143)

#define NC_MAX_ACCELERATION_OUT_OF_RANGE   (-144)

#define NC_MAX_DECCELERATION_OUT_OF_RANGE   (-145)

#define NC_MAX_JERK_OUT_OF_RANGE   (-146)

#define NC_DIRECTION_TYPE_OUT_OF_RANGE   (-147)

#define NC_NEGATIVE_VELOCITY   (-148)

#define NC_COORD_SYSTEM_TYPE_OUT_OF_RANGE   (-149)

#define NC_CIRCLE_MODE_OUT_OF_RANGE   (-150)

#define NC_PATH_CHOICE_TYPE_OUT_OF_RANGE   (-151)

#define NC_ARC_SHORT_LONG_TYPE_OUT_OF_RANGE   (-152)

4.6   Warnings instead of Errors

In the current version, the following are no longer considered as errors, but warnings:

- PowerOn command when already on.
- PowerOff command when already off.
- ChangeOperationMode when already in the mode.
- GroupEnable when group is enabled
- GroupDisable when group is already disabled.
- Zero length group profiles

Release Notes - New Firmware Version for GMAS.

Version 1.0.3.0

The "***uImage_v1.0.3.0_2011_08_09.gms***" is a new firmware release for the GMAS Master Motion Controller, which support the following new major features:

5. General

1. Updated Profiler.
2. First Ethercat Release supporting following modes:
    - Preoperational and Operational mode support
    - Ethercat Configuration mode support.
    - Cyclic Position.
    - Profile Position.
    - Homing.
    - Touch Probe functionalities.
    - Beckhoff Ethercat IO Support
2. MoveRepetitive Support
3. Synchronized Timer support.
4. Virtual Encoder support
5. EoE.
3. 1D, 2D and 3D Dynamic Error Correction.
4. Data recording of additional parameters:
    - Digital Outputs
    - Digital Inputs
    - Drive Position Error
    - Actual drive Hardware position (before Error Correction).
    - Error Correction Value.
5. CPP Library as part of the MMC_LIB release supporting all of the MMC_LIB functionalities

5.1   Updated Profiler

In this version, we introduced an automated profiler tester that generated and tested thousands of different profiles at different cycle times. Many flaws were fixed and references were created.

5.2   First Ethercat Release

This is the first released Ethercat version, supporting the following features:

- Preoperational and Operational mode support – There is an ability to transfer between operational mode and preoperational mode. This is performed by calling the newly added APIs:
    1. MMC_ChangeToPreOPMode
    2. MMC_ChangeToOperationMode

  In operational mode, the user can perform all motions via the GMAS, but cannot perform motions / tune the drive on the drive level. However, of course, the user may view, record variables on the drive level.

  In preoperational mode, the user can perform motions on the drive level, without the GMAS update the cyclic process data.

- Ethercat Configuration mode support – In this mode, the user can open the EAS and view, scan, start the Ethercat network. This is performed by calling the following API's
    1. MMC_EnableEthercatConfigMode
    2. MMC_DisableEthercatConfigMode
- Profile Position DS402 mode. Same functionalities as CAN are supported.
- Homing – all DS402 homing modes are supported.
- Touch Probe functionalities (Ethercat Only) – After configuring the following parameters to be updated via the Ethercat process data, callbacks are sent from the GMAS with the position of the Touch Probe (compare position):

    1. Touch probe status (Input)
    2. Touch probe pos1 pos value (Input)
    3. Touch probe pos1 neg value (Input)
    4. Touch Probe Function (Outputs)

  The MMC_TouchProbeEnable API is to be called in order to Enable the touch probe functionality. In addition, the XXX callback type is to be configured.

- Beckhoff Ethercat IO Support – The GMAS supports basic Beckhoff Ethercat IO's:
    1. Digital Inputs / Outputs.
    2. Analog Inputs / Outputs.

  Callbacks stating that a digital input changed (similar to CAN) is also supported. The following 'C' functions (and compatible C++ functions) were added:

1. MMC_ECATIOEnableDIChangedEvent
2. MMC_ECATIODisableDIChangedEvent
3. MMC_ECATIOReadDigitalInput
4. MMC_ECATIOWriteDigitalOutput
5. MMC_ECATIOReadAnalogInput
6. MMC_ECATIOWriteAnalogOutput

5.3   Move Repetitive Support

New API's were added in order to perform repetitive motions in single axis:

- MMC_MoveAbsoluteRepetitiveCmd
- MMC_MoveRelativeRepetitiveCmd

And in Group axis:

- MMC_MoveLinearAbsoluteRepetitiveCmd
- MMC_MoveLinearRelativeRepetitiveCmd

Basically, the user can perform reciprocated motions between two points. Either single or group motions. Absolute or Relative. This version introduced the repetitive motion in cyclic position and interpolated position DS402 motion modes only.

5.4   Synchronized Timer Support

In order to promise that the user's program within the GMAS is synchronized to the main GMAS PLCopen cycle, the following API was added:

MMC_CreateSYNCTimer

Using this function, the user can define every n (n >=1) GMAS cycles to call a defined callback function. This callback is called only if the previous callback finished before the time to call the callback function again.

In order to close the callback, the following API was added:

MMC_DestroySYNCTimer

5.5   Virtual Encoder Support

In order to synchronize a number of axes via CAN, and perform ECAM motions on the drive level, the user must purchase an expensive CAN encoder supporting DS406. Instead, the same functionality can be emulated via the GMAS. The GMAS can send a broadcast CAN message (actually a group message) containing the position of one of the GMAS axes. The position may be either the target position or the actual position of the axis. In addition, the axis may be a virtual axis on the GMAS.

5.6   EoE

EoE – Ethernet over Ethercat was finally introduced in this version. Using EoE, the GMAS acts as a 'switch' to the Ethercat network, and transparently transfers data to the Ethercat network sent over UDP to the drives(Either from a host (EAS for example) or via the GMAS user program). Using EoE, the EAS communicates with the drives, via the GMAS as a gateway

The drives must be configured, using the Ethercat Configuration tool, to support EoE. The IP address and MAC addresses must be configured via the Ethercat Configuration tool together with defining the list of connections in the EAS.

There is no 'C' API to communicate via EoE. We introduced a CPP class – CMMCUDP that handles basic asynchronous UDP communication. No necessarily even EoE.

5.7   1D, 2D and 3D Dynamic Error Correction

Error mapping is required for correction of non-linear mechanical position errors. The correction is done by taking into account errors that have been pre-defined by the user at

---

some discrete position points (measurement grid points), and altering the actual position readings.

The meaning of 2-D Error correction is, that every correction point (x,y) on a two dimensional grid will be defined as a function of any two axes positions. The point actually defines an error for a specific axis. For instance, axis Z correction may be a function of X and Y. X correction, can be a function of X itself and Y position.

CorrectedPosition=HardwareReading+ErrorCorrection.

…where, *ErrorCorrection* is a function of two position inputs.

3-D error correction is, in a way, is similar to 2-D error correction. In 2-D error correction, a 2-D table and grid is defined for any specific 2 axes. The idea of 3-D error correction is having numerous of these grids, "one above the other". As defined for the 2-D mode, the third axis may also be defined as any of the axes, and this must be defined by the user in advance of course (as part of the general setup).

Operation Principles:

- An error correction table is loaded to user-kernel shared memory and enabled
- The Kernel module performs the error correction algorithm on all enabled error correction tables. An error correction offset is calculated.
- The corrected position is sent to the drive (based on target position + correction offset) and the actual position is updated accordingly as well.
- A total of 4 error correction tables can be loaded to the memory.
- Every table can be enabled or disabled (The process of loading a table can take a long time (large table, etc.), and there is a possibility that the user would like to toggle the table on and off. For this purpose, we differ between "loaded" and "enabled" status. When the table is loaded, it is still not enabled – not ready for use. If the table is disabled – that means the table is still present in memory, but not available for use)
- 100000 points are available for the usage of the 4 tables. For example, if the first table contains 78000 points, the second and the third 10000, and we will try to insert fourth with 5000 points – we will fail. The user is responsible for handling the memory start and offsets.
- The error correction table is actually a file, with the following format:

Table size	25
Start offset	0
Error table dimension	2
Start position	20000 20000
Axis grid size	16 16
Table dimensions	5 5

Table #1 Start Data

0	100	0	100	0
0	100	0	100	0
0	100	0	100	0

<table>
<tr><td>0</td><td>100</td><td>0</td><td>100</td><td>0</td></tr>
<tr><td>0</td><td>100</td><td>0</td><td>100</td><td>0</td></tr>
</table>

Table #1 End Data

The file can be edited via excel and then download to the GMAS.

- The following API's were added to support the Error mapping:

MMC_LoadErrorCorrTableCmd

MMC_UnloadErrorCorrTableCmd

MMC_EnableErrorCorrTableCmd

MMC_DisableErrorCorrTableCmd

In addition, the following recording variables were added:

-    Actual HW Pos – The position value reported by the drive.
-    Error Correction – Error correction delta for the specific axis.

5.8   Data Recording of Additional Parameters

In this version we added the ability to record the following parameters:

-    Digital Outputs
-    Digital Inputs
-    Drive Position Error
-    Actual drive Hardware position (before Error Correction).
-    Error Correction Value.

Release Notes - New Firmware Version for GMAS.

Version 1.0.2.0 and Version 207 for MMC_LIB

The "*uImage_v1.0.2.0_2011_02_01.gms*" is a new firmware release for the GMAS Master Motion Controller, which support the following new major features:

1. General

1. Bug fixes to binary Interpreter support.
2. OpenIPC – New function that opens an explicit IPC connection, and closes latter connections if they exist.
3. Data recording of ACDC now supported.
4. Bug fixes regarding the connection handling and Modbus listener threads.
5. Support for Preoperational modes for Ethercat and CANBus and exiting back to normal mode.
6. Support for Ethercat Configuration mode, and exiting back to normal mode.
7. Support for Event based PDOs. Please refer to function ConfigPDO3, ConfigPDO4 (User and Regular).
8. Many bug fixes.

Date: 29/12/2010

Revision: 1.0

Release Notes - New Firmware Version for GMAS.

Version 1.0.1.8 and Version 205 for MMC_LIB

2. General

The "*uImage_v1.0.1.8_2010_12_21.gms*" is a new firmware release for the GMAS Master Motion Controller, which support the following new major features:

1. Fast responses to events from drive.
2. Binary Interpreter support.
3. Multiple NIC from PC support – now in library.
4. Many new API's.
5. Version supporting integrative Eclipse debugging and IDE.
6. Many bug fixes.

2.1 Fast Responses to events from Drive

The user will be able to configure each axis PDO3/4 event mechanism to the following:

- PDO mapping, without notification to the user.

- PDO mapping, with cycle time basis notification to the user.

- PDO mapping, with immediate notification to the user.

The PDO3/4 Received Event mode, could be configured via the ConfigEventModePDO3(), ConfigEventModePDO4() APIs respectively.

The API will accept as argument one of 3 event modes:

- 0 – No notification to user.

- 1 – Cycle time basis notification to user.

- 2 – Immediate notification to user.

Communication events mechanism
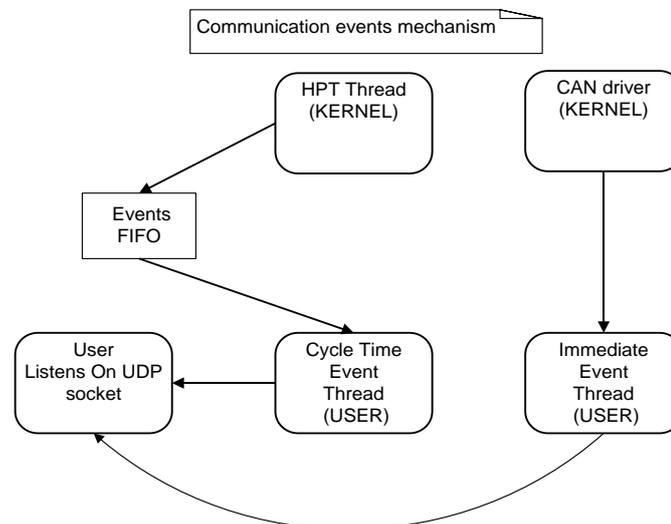
2.2 Binary interpreter

The GMAS API supports the following:

- *ElmoGetParameter()*–e.g. *MMC_ElmoGetParameter(gConnHndl,gAxisRef[i],"AC",0) ;*

- *ElmoSetParameter() – e.g.*
  MMC_ElmoSetParameterWrapper(gConnHndl,gAxisRef[2],"MO",0,&iValue);

- *ElmoGetArray() - e.g. ElmoGetArray(axisref, "et", index)*

- *ElmoSetArray() - e.g. ElmoSetArray(axisref, "et", index, val)*

- *ElmoCall() - e.g. ElmoCall(axisref, "kI")*

- *ElmoExecuteLabel() - e.g. ElmoExecuteLabel(axisref, "home_1")*

The above functions will be downloaded to the DRIVE via *Bin Interpreter* or *OS Interpreter* mechanism (The only function which will use the *OS interpreter* mechanism is: *ElmoExecuteLabel()*, because in this case, we may need more than 8 bytes message for performing the *execute program* command).

The above functions could be used via RPC or IPC connection.

The Get functions could be performed in 2 ways:

- **Synchronously** – the function will not return, until response is accepted from the DRIVE. The returned function will return the response from the DRIVE.

- **Asynchronously** – the function will return immediately without waiting for response from the DRIVE. In the GMAS, when the response from the DRIVE will be accepted, it will be sent via a UDP message to the library, the library *connection listener* thread will process the message and if it is a *Binary Interpreter – Get command*, it will be processed as follows:

  - Checking if the *Query operation* mode is set to *asynchronous query* mode. If it is the mode:

    The uploaded data is kept in a per axis FIFO – (the *asynchronous query* FIFO) and the axis *asynchronous query* FIFO index is incremented.

    If the *Query operation* mode is set to *synchronous query* mode, the response data will be copied to the user out structure parameter.

  The user will be able to access the axis *asynchronous query* FIFO, whenever he chooses and retrieves data from the axis FIFO according to the axis FIFO index progress, which its value he will be able to monitor via a specific API.

The data returned from the Get functions, could be long/float, therefore, storing the uploaded data by the library in the axes FIFOs and enabling retrieve of the data by the user, should be done, in attention to the specific data type which is taking care.

Notes:

- The set/call function receives a parameter stating whether the parameter is long/float.

- In case of asynchronous get operation, a FIFO with length 1 will be managed for each axis.

- If an error was sent from the DRIVE as a result of sending Interpreter command, the reactor will be emptied and an error callback will be sent to the library. The drive in this case does NOT enter the Error Stop state.

Get function – Asynchronous Mode

In order to work in this mode, the user will call the regular asynchronous functions.

The FIFO index for each axis, could be monitored by the user, via the *ElmoQueryOperationFIFOIndex()* API.

The user may retrieve data from the axis *Query Operation* FIFO at location *index* in any time, via the *ElmoQueryOperationFIFORetrieveData(index)* API.

Get function – Synchronous Mode

In order to retrieve data in a synchronous manner, the *MMC_ElmoGetArrayAndRetrieveData*

and the *MMC_ElmoGetParameterAndRetrieveData* should be called.

For a full list of API's, please refer to XXX for API list.

2.3 Support for Multiple NIC in PC

As per previous GMAS version, there was a problem when trying to download, upload files when the host computer had several NIC's.

A new API was created: MMC_RpcInitConnection. This API has an additional parameter which states the IP of the host, to be used.

2.4 New APIs

The following new API's were added to this version. Please refer to the API manual for further information.

- MMC_ElmoSetParameter
- MMC_ElmoGetParameter
- MMC_ElmoGetArray
- MMC_ElmoSetArray
- MMC_ElmoGetParameterAndRetrieveData
- MMC_ElmoGetArrayAndRetrieveData
- MMC_ElmoQueryOperationFIFOIndex
- MMC_ElmoQueryOperationFIFORetrieveData
- MMC_ElmoQueryOperationFIFOIndexReset
- MMC_ElmoExecuteLabel
- MMC_ElmoCall
- MMC_CfgEventModePDO3Cmd
- MMC_CfgEventModePDO4Cmd
- MMC_GetCommDiagnostics
- MMC_ResetCommDiagnostics
- MMC_GetRactorStatistics
- MMC_SetHeartBeatConsumerCmd
- MMC_ReadDigitalOutputs32Bit
- MMC_WriteDigitalOutputs32Bit
- MMC_MbusStoptServer changed to MMC_MbusStopServer

2.5 Version Supporting Integrative Eclipse Debugging and IDE

This version includes numerous scripts that are located within the GMAS (/mnt/jffs/usr) that are called by the Eclipse environment.

2.6 Many Bug Fixes

Release Notes - New Firmware Version for GMAS.

Version 1.0.1.4

3. General

The "**uImage_v1.0.1.4_2010_10_11.gms**" is a new firmware release for the GMAS Master Motion Controller, which support the following new major features:

1. Support Dig output in standard PDO settings if supported by drive (new Elmo Digital Output object).
2. Emergency handling.
3. FULL Error handling support as per PLCopen.
4. Error recovery.
5. Ability to map UI and UF and send as output PDO's
6. New default PDO mapping abilities.
7. DS401 axis type – Digital Outputs, Digital Inputs.
8. DS301 axis type.
9. DS406 axis type.
10. New data recording features.
11. Many bug fixes.
12. Motion abortion as per PLCopen.
13. Ability to send SDO's from user programs and from a RPC host.
14. USB support for basic system configuration.
15. New U-Boot supporting enhanced version update mechanism including auxiliary backup image.
16. Enhanced events mechanism: Ability to receive events upon the receiving of Modbus registers from Modbus host and from DS401 Digital Input updates.
17. Function attributes stating the function call context.
18. New resource file definitions.
19. New library APIs and interfaces to the GMAS in order to support all of the above.

In addition to these new major features, this firmware version also supports some other new features, including modifications in the current API:

- UserCfgPDO3 and UserCfgPDO4 now receive a parameter stating whether the PDO configuration is PDO TX or PDO RX.

- Interfaces to CfgPDO3 and CfgPDO4 (user and general) changed due to new types of UI, UF supported.

- PDO RX (sent by GMAS) has 5 groups of PDOs that can be set

- SendSDO - download - API is now synchronous (on the API level).

- Interface to CAN Rawdata that is send via UDP channel was changed due to alignment problems. This version will not work with composer. Needs to be recompiled with new library.

4. Description of New Features Supported by this Release

4.1 Support Digital Output in Standard PDO settings

DS402 as a standard does not have a Digital Output object (It does support a digital input object). A new SimplIQ version was released *SimplIQ 2.02.09.13 20May2010.abs.* This version supports object 0x2210.

The object sets and gets general purpose digital output. The object behaves similar to the low byte of OP commands (bits 0-7).

General purpose must be first defined using OL[] command.

Command to digital output which were not defined as general purpose shall be discarded. No indication is reflected to the user.

When object is read, the value reflects the following:

Bits 0 - 6: General purpose setting

Bit 7: AOK state. Relevant if one of the outputs is configured to AOK function.

Object description:

Index	2210h
Name	Digital output
Object code	VAR
Data type	UNSIGNED8
Category	Mandatory

Entry description:

Access	R\W
PDO mapping	Yes
Value range	UNSIGNED8

Default value	0

As you can see, this object is an eight bit object. The user may select via the EAS interface whether the drive supports this feature or not (and of course – that the user need Digital Output support).

If the user selected to use Digital Output, and the drive does not support this (the drive version is prior to the version mentioned above), then the axis will not operate as the PDO mapping will fail. The error the user will receive (as part of the Error mechanism) will be a communication / mapping error to the drive.
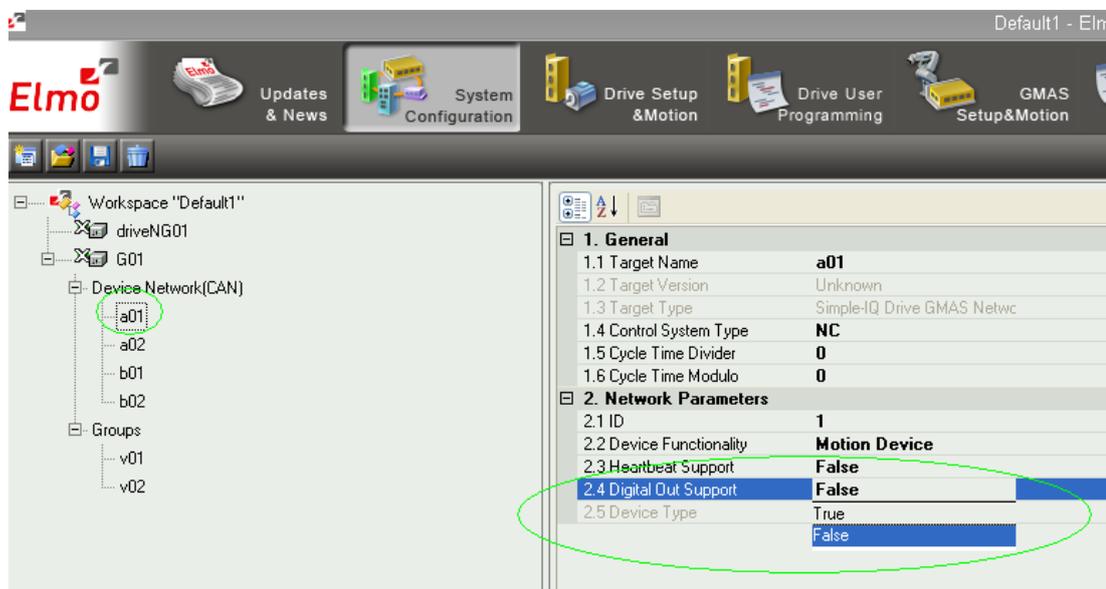
Changing the **Digital Out Support** field in the EAS (as illustrated above) affects the following resource file parameter, under Motion Devices cluster:

<NODE NAME="a02" TYPE="ds402" SUBTYPE="ELMO" ID="2" NC_AXIS="1" MODULO="0" DIVIDER="0" DOUT_SUPPORT="0"

This object is added to PDO1 (7 bytes are sent instead of 6 per RX PDO1), and is sent according to the current motion mode of the drive.

For instance: If the drive is in interpolated motion mode – the object will be sent via PDO1 every SYNC. If the drive is in Profile Position – then the object is sent every "Onchange", and so on...

Changing and reading the Status of Digital Outputs

The following PLCopen functions are to be used in order to Read / Write digital Outputs:

- o ReadDigitalOutputs

- o WriteDigitalOutputs

4.2 Emergency Handling

The DS301 emergency objects are of course supported by the GMAS.

The FULL emergency message (Error Code + Error Register + Manufacturer Specific), if received, may be obtained from the AxisErrorCode.

Once an emergency is received, the axis enters the error stop state.

4.3 FULL Error handling support as per PLCopen

The errors according to categories:

- Catastrophic Errors. These are errors that occurred as part of the system, and are unrecoverable.

    - Cannot open sockets.

    - General socket errors.

    - Memory allocation errors.

    - Ethercat Errors

        - Working counters.

        - Disconnection of specific node.

    - IPC semaphore errors.

    - File system errors.

- System Errors.

    - CAN overflow error – this means we missed a specific message. We do not know what we missed. Can be catastrophic.

    - Invalid / no resource file.

- Communication errors to drive

    - Heartbeat consumer error.

    - Mapping SDO's

- Emergency – communication related.

- Motion errors

- Fault bit / Emergencies from drive

- Profiler errors.

- Change of operation mode failures.

The errors mentioned above are in addition to the errors already reported as part of the PLCopen function execution.

4.4 Error Recovery

What are we to do once we encounter an error?

As per PLCopen definitions, the ErrorState is to be entered once encountering such an error. Once entering the ErrorState there is to be an attempt to stop the motion of the specific axis. As per PLCopen:

The intention of the "ErrorStop" state is that the axis goes to a stop, if possible. There are no further FBs accepted until a reset has been done from the ErrorStop state

Therefore – the Control Word of the axis in the error state is to be forced to '0' – thus causing a halt to the axis. This is to be done for ALL error categories. In order to exit the ErrorState – the MC_Reset function is to be called.

The errors are handled differently per category. The following lists the handling, per category:

Catastrophic Error Handling

- This is an unrecoverable state. The MultiAxisControl application will be closed immediately. The system must be reset.

System Error Handling

- An attempt is to be made to stop all axes in the system this may not be possible due to CAN overflow, Ethercat WC's, etc …

- Once such an error occurs – ALL axes move to the ErrorState. the real-time no longer sends data.

- The AxisError is set accordingly.

- This is an unrecoverable state. The system must be reset. Files may be downloaded (new resource files, etc …).

Communication Error Handling

- An attempt is to be made to stop the problematic axis.

- The node is to be flagged to be in *ERROR STOP* state, all axis communication parameters will be initialized, from this point onwards, the real-time no longer sends / prepares data for such a node.

- The AxisError is set accordingly.

- The only way to recover from such a problem is by the MC_Reset function. This function will automatically reinitialize the axis, as if it were at power up. – (The axis is to be remapped to a default motion mode – in NC mode to Interpolated Position mode, in Non NC mode to Profile Position mode, the Heartbeat mechanism is initialized…etc).

Heartbeat Error Handling

- An attempt is to be made to stop the problematic axis.

- The node is to be flagged to be in *ERROR STOP* state, all axis communication parameters will be initialized, from this point onwards, the real-time no longer sends / prepares data for such a node.

- The AxisError is set accordingly.

- The only way to recover from such a problem is by the MC_Reset function. This function will reset axis FB's list, and set axis to DISABLED state. In respect to communication, the axis will stay in the last motion mode, the axis is not go to initial communication state.

Emergency Message Error Handling

- An attempt is to be made to stop the problematic axis.

- The node is to be flagged to be in *ERROR STOP* state, all axis communication parameters will be initialized, from this point onwards, the real-time no longer sends / prepares data for such a node(If the *Fault* bit in the status word is set, we repeatedly sends control word with *Clear Fault* bit set until the status word *Fault* bit is cleared).

- The AxisError is set accordingly.

- The only way to recover from such a problem is by the MC_Reset function. This function will reset axis FB's list, and set axis to DISABLED state. In respect to communication, the axis will stay in the last motion mode, the axis is not go to initial communication state.

New Node on the bus

- When axis appears on the Can bus, after the initial power-up scan process, power-up communication initializations are done and the axis is set to *ERROR STOP* state, from this point onwards, the real-time no longer sends / prepares data for such a node. This is done for synchronizing between DS402 state and PLCopen state.

- The only way to recover from such a problem is by the MC_Reset function. This function will reset axis FB's list, and set axis to DISABLED state.

Note:

After the MC_Reset function was invoked, the axis will return to normal mode, **only** if the Fault bit in the status word is cleared, no emergency message is pending, no communication error exists and no heartbeat error exists.

The user must check all the error classes status bits before performing the MC_Reset function.

For example, no need to perform useless MC_Reset invocation if the axis is still in Heartbeat error.

4.5 Ability to map UI and UF and send as output PDO's from the GMAS

The drives UI and UF parameters can now be mapped to RX PDOs (PDOs sent by the GMAS to the drive). These UI variables are configured by calling the following API:

MMC_LIB_API int MMC_CfgUserParamEvPDO3Cmd(

      IN MMC_CONNECT_HNDL hConn,

      IN MMC_AXIS_REF_HNDL hAxisRef,

      IN MMC_CONFIGUSERPARAMEVENTPDO3_IN* pInParam,

      OUT MMC_CONFIGUSERPARAMEVENTPDO3_OUT* pOutParam);

With the following structures:

Input Structure


```
typedef struct
{
        unsigned short usEventTimer;
        unsigned char ucEventGroup;
        unsigned char ucSubIndex;
        unsigned char ucPDOCommParam;
        unsigned char ucPDOType;
}MMC_CONFIGUSERPARAMEVENTPDO3_IN;
```


usEventTimer – States, in case the ucPDOCommParam is NOT *OnSync* – the time, in ms, the PDO is to be updated. Should remain 0 if the user wants an update *OnChange*.

ucSubIndex – Index of UI / UF to be set. The consecutive index will be used for the second defined UI.

ucEventGroup - Can be of the following types:

        NC_COMM_RXEVENT_GROUP1,   /// User Integer 3+ user integer 3 aux.

        NC_COMM_RXEVENT_GROUP2,   /// User Integer 4+ user integer4 aux.

        NC_COMM_RXEVENT_GROUP3,   /// User Float 3   + user Float 3 aux.

        NC_COMM_RXEVENT_GROUP4,   /// User Float 4   + user Float 4 aux.

        NC_COMM_RXEVENT_GROUP5   /// User Integer 3+ user Float 3.

ucPDOCommParam – 1 or 255. 1 states that the data is updated OnSync. 255 states that the data is updated *OnChange*.

ucPDOType – 0 = RX PDO. 1 = TX PDO.

For example:

If I defined `cfg_pdo3_in` as type CONFIGUSERPARAMEVENTPDO3_IN and then

```
cfg_pdo3_in.ucEventGroup      = NC_COMM_RXEVENT_GROUP1;
cfg_pdo3_in.ucPDOCommParam    = 1;  // on sync
cfg_pdo3_in.ucSubIndex        = 5;
cfg_pdo3_in.usEventTimer      = 0;  // Irrelevant when on sync
cfg_pdo3_in.ucPDOType         = 0 ; // RX PDO
```

…then the data that represents the UI in the GMAS, will be sent, OnSync, to the drive. The data will be mapped to UI[5] and UI[6].

In order to write the data to the Output UI data in the GMAS, the *MMC_WriteBoolParameter* function must be called. The function receives, as a parameter, the relevant param number to write to. The enumerations are as follows:

```
MMC_I_COMM_EV_USR_1_PARAM      = 33,          /// PDO TX From Drive

MMC_I_COMM_EV_USR_1_AUX_PARAM= 34,            /// PDO TX From Drive

MMC_F_COMM_EV_USR_1_PARAM      = 35,          /// PDO TX From Drive

MMC_F_COMM_EV_USR_1_AUX_PARAM= 36,            /// PDO TX From Drive

MMC_I_COMM_EV_USR_2_PARAM      = 37,          /// PDO TX From Drive

MMC_I_COMM_EV_USR_2_AUX_PARAM= 38,            /// PDO TX From Drive

MMC_F_COMM_EV_USR_2_PARAM      = 39,          /// PDO TX From Drive

MMC_F_COMM_EV_USR_2_AUX_PARAM= 40,            /// PDO TX From Drive

MMC_I_COMM_EV_USR_3_PARAM      = 41,          /// PDO TX TO Drive

MMC_I_COMM_EV_USR_3_AUX_PARAM= 42,            /// PDO TX TO Drive

MMC_F_COMM_EV_USR_3_PARAM      = 43,          /// PDO TX TO Drive

MMC_F_COMM_EV_USR_3_AUX_PARAM= 44,            /// PDO TX TO Drive

MMC_I_COMM_EV_USR_4_PARAM      = 45,          /// PDO TX TO Drive

MMC_I_COMM_EV_USR_4_AUX_PARAM= 46,            /// PDO TX TO Drive

MMC_F_COMM_EV_USR_4_PARAM      = 47,          /// PDO TX TO Drive

MMC_F_COMM_EV_USR_4_AUX_PARAM= 48,            /// PDO TX TO Drive
```

Since the `cfg_pdo3_in.ucEventGroup     = NC_COMM_RXEVENT_GROUP1` was set for the PDO mapping – in order to write to UI[5] in the drive `MMC_I_COMM_EV_USR_3_PARAM` should be selected (as it is the 1st UI for PDO TX).

4.6 New Default PDO3 and PDO4 Mapping Options

Additional PDO mapping were added. Firstly – there is an ability (as per previous section) to send PDOs from the GMAS. Currently – only the following are supported as RX PDOs:

NC_COMM_RXEVENT_GROUP1, /// User Integer 3   + user integer 3 aux.

NC_COMM_RXEVENT_GROUP2, /// User Integer 4   + user integer4 aux.

NC_COMM_RXEVENT_GROUP3, /// User Float 3   + user Float 3 aux.

NC_COMM_RXEVENT_GROUP4, /// User Float 4   + user Float 4 aux.

NC_COMM_RXEVENT_GROUP5 /// User Integer 3   + user Float 3.

Also – there are additional TX PDOs (PDOs sent from the drive to the GMAS). The full list is as follows:

```
typedef enum
{
        NC_COMM_EVENT_NO_GROUP = 0,

        NC_COMM_EVENT_GROUP1,              /// Position Actual Value + Velocity Actual Value

        NC_COMM_EVENT_GROUP2,     /// Position Actual Value + Digital Inputs

        NC_COMM_EVENT_GROUP3,    /// Digital Inputs + Velocity Actual Value

        NC_COMM_EVENT_GROUP4,    /// Digital Inputs + Current Actual Value

        NC_COMM_EVENT_GROUP5,     /// Current Actual Value + Position Actual Value

        NC_COMM_EVENT_GROUP6,     /// Current Actual Value + Velocity Actual Value

        NC_COMM_EVENT_GROUP7,              /// User parameter(Integer)+Actual Pos

        NC_COMM_EVENT_GROUP8,              /// User parameter(Integer)+Digital Inputs

        NC_COMM_EVENT_GROUP9,              /// User parameter(Float)+Actual Pos

        NC_COMM_EVENT_GROUP10,   /// User parameter(Float)+Digital Inputs

        NC_COMM_EVENT_GROUP11,    /// Digital Inputs

        NC_COMM_EVENT_GROUP12,   /// User parameter integer 1 + User parameter integer 2.

        NC_COMM_EVENT_GROUP13,   /// User parameter integer 3 + User parameter integer 4.

        NC_COMM_EVENT_GROUP14,   /// User parameter float   1 + User parameter float 2.

        NC_COMM_EVENT_GROUP15   /// User parameter float   3 + User parameter float 4.

}NC_COMM_EVENT_GROUP;
```

In order to update these parameters, the *ucEventGroup* parameter in the CfgUserParamEvPDO3Cmd, CfgUserParamEvPDO4Cmd, CfgRegParamEvPDO3Cmd, CfgRegParamEvPDO4Cmd functions must be set.

For instance, if we were to update, *OnSync*, *actual position + velocity* then the following is to be sent:

```
cfg_pdo3_in.ucEventGroup      = NC_COMM_EVENT_GROUP1;

cfg_pdo3_in.ucPDOCommParam    = 1;

cfg_pdo3_in.usEventTimer      = 1;  // get data every sync time
```

If we were to update, *OnSync, User parameter(Float UF[1])+ User parameter(Float UF[2])* then the following is to be sent:

```
cfg_pdo4_in.ucEventGroup      = NC_COMM_EVENT_GROUP14;

cfg_pdo4_in.ucPDOCommParam    = 1;  // on sync

cfg_pdo4_in.ucSubIndex        = 1;

cfg_pdo4_in.usEventTimer      = 1;  // on sync

cfg_pdo4_in.ucPDOType         = 1 ; // TX PDO
```

4.7 DS301 Axis Type

An additional DS301 *device type* was added to the GMAS. This device type allows sending standard SDO's to a generic DS301 based module.

In order to select the DS301device via the EAS, the *Device Functionality* should be of type *General Device* and the *Device Type* should be *DS301* as per illustration below:
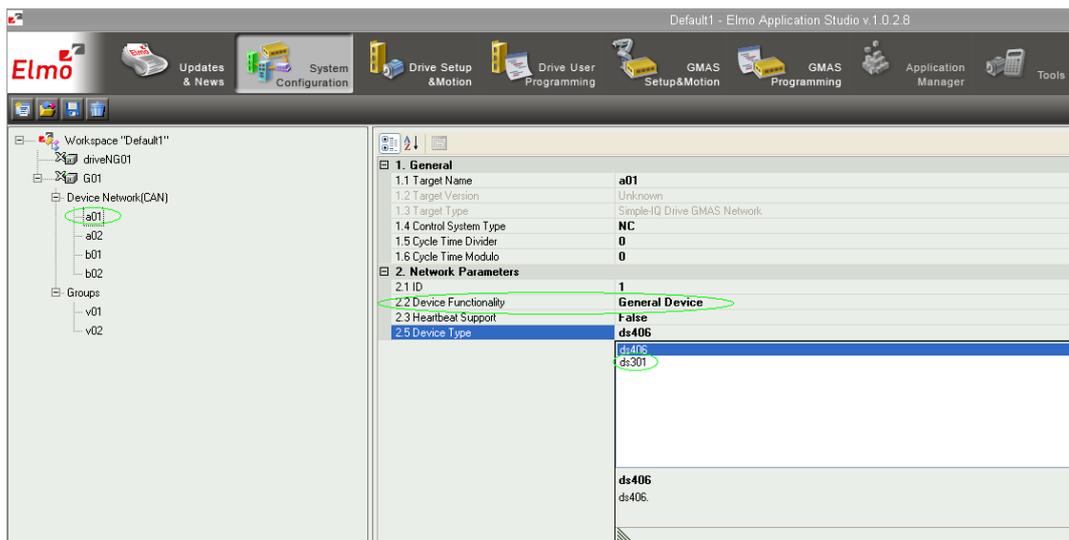
The user may also choose whether to support theDS301 heartbeat mechanism (GMAS as consumer). This can be set by selecting the *Heartbeat Support* options.

Calling DS301 devices from within the code is identical to any motion related device. An *AxisRef* must be obtained in the identical way.

Only some of the functions support DS301.

The resource file has a new cluster by the name of <GENERAL_DEVICES>. This cluster also holds DS301 related information. For instance, if I were to define a DS301 device that supports the DS301 heartbeat mechanism, it will look as follows:

        <GENERAL_DEVICES>

                <NODE NAME="a01" TYPE="ds301" ID="1" HBT_SUPPORT="1"/>

        </GENERAL_DEVICES>

## 4.8 DS401 Axis Type – Digital Outputs, Digital Inputs

The DS401 device type is now supported in the GMAS. Currently, only Digital Inputs

And Digital Outputs modules are supported. Analog Inputs and Analog Outputs are not yet supported in this version.

The DS401 standard includes many optional objects that may be supported by DS401 CAN IO Vendors. The GMAS supports the essential objects only (and not what is defined by the DS401 standard as *optional*. For instance – the Filter and Polarity objects are not automatically sent by the GMAS. If the user would like to set specific DS401 parameters, this of course may be done, by sending the specific SDO messages to the module.

A DS401 device may include either Digital Inputs, Digital Outputs or both (and in the future – Analog Inputs/Outputs or all). In order to select the DS301device via the EAS, the *Device Functionality* should be of type *IO Device* and the *Module Type* should be

Selected (either Digital Inputs, Digital Outputs or both).
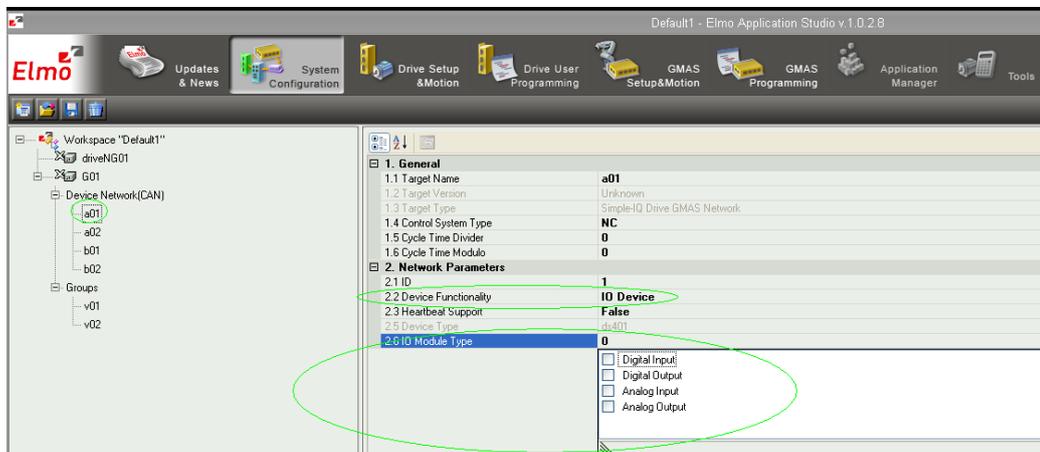
Since only DS401 default values are supported, the update type of the Digital Inputs / Outputs are on change. The user may, if supported by the module, modify this to *OnSync* by setting the relevant SDOs.

Only 64 Digital Inputs / Outputs are supported per CAN ID. If a DS401 CAN module has more than 64 IOs, they may be read via SDOs.

The resource file has a new cluster by the name of < IO_DEVICES>. This cluster also holds DS401 related information. For instance, if I were to define a DS401 device that supports the DS301 heartbeat mechanism, it will look as follows:

```
<IO_DEVICES>
                <NODE NAME="a03"  TYPE="ds401" ID="3" IO_MODULE_TYPE="0" HBT_SUPPORT="1" />
                <NODE NAME="a04"  TYPE="ds401" ID="4" IO_MODULE_TYPE="1" HBT_SUPPORT="1" />
                <NODE NAME="a05"  TYPE="ds401" ID="5" IO_MODULE_TYPE="2" HBT_SUPPORT="1" />
</IO_DEVICES>
```

Where:

ID #3 is a digital input module.

ID #4 is a digital output module.

ID #5 is a digital input and digital output module.

The MODULE_TYPE definition is as follows:

Bit 0: D. Input.

Bit 1: D. Output.

Future: Bit 2: A. Input.

Future: Bit 3: A. Output

The GMAS automatically identifies the amount and types of IOs on the network. This is verified (the type of IO) at powerup of the GMAS.

DS401 API's

In order to read DS401 Digital Inputs, the following API was added:

```
MMC_LIB_API int MMC_ReadDS401DInput(

        IN MMC_CONNECT_HNDL hConn,

        IN MMC_AXIS_REF_HNDL hAxisRef,

        IN MMC_READDI_IN* pInParam,

        OUT MMC_READDI_OUT* pOutParam) ;
```

With the following structures:

Output Structure

```
typedef struct mmc_readdi_out

{

#ifdef WIN32

        unsigned __int64 ulliDI;

#else

        u_int64_t         ulliDI;

#endif

        unsigned short usStatus;

        unsigned short usErrorID;

}MMC_READDI_OUT;
```

An example for using this function:

```
MMC_READDI_IN        ri ;

MMC_READDI_OUT       ro ;
```

MMC_ReadDS401DInput(conn_hndl,3,&ri,&ro) ;

The value of the Input resides in ro.ulliDI.

In order to write DS401 Digital Outputs, the following API was added:

MMC_LIB_API int MMC_WriteDS401DOutput(

IN MMC_CONNECT_HNDL hConn,

IN MMC_AXIS_REF_HNDL hAxisRef,

IN MMC_WRITEDO_IN* pInParam,

OUT MMC_WRITEDO_OUT* pOutParam) ;

With the following structures:

Input Structure

```
typedef struct mmc_writedo_in
{
#ifdef WIN32
        unsigned __int64 ulliDO;
#else
        u_int64_t          ulliDO;
#endif
}MMC_WRITEDO_IN;
```

An example for using this function:

MMC_WRITEDO_IN   wi ;

```
MMC_WRITEDO_OUT        wo ;


wi.ulliDO = 0x5a ;

MMC_WriteDS401DOutput(conn_hndl,2,&wi,&wo) ;
```
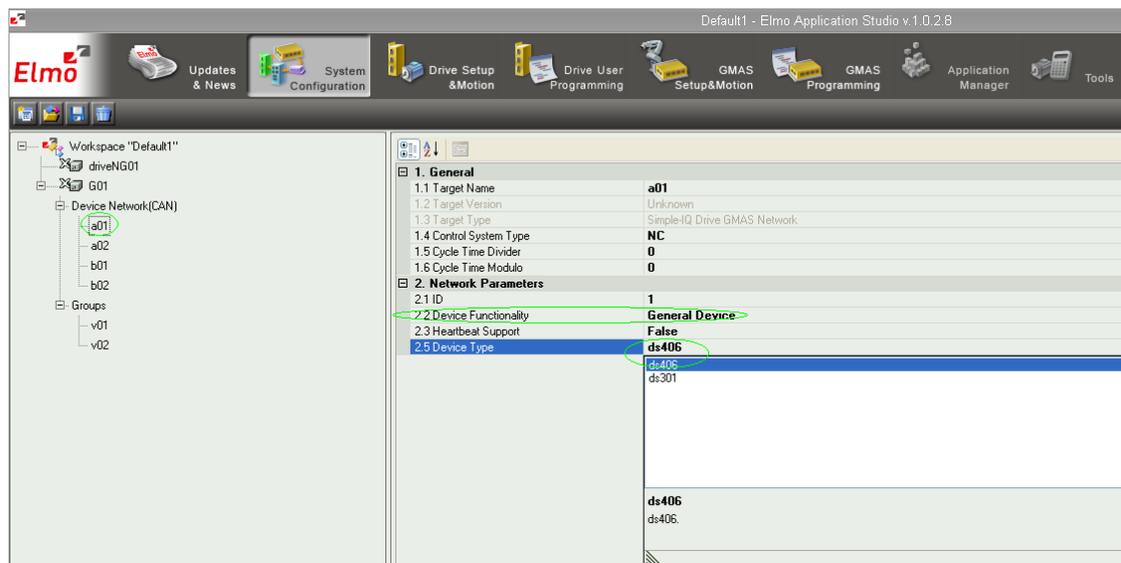
4.9 DS406 Axis Type

An additional DS406 device type (CAN encoder) was added to the GMAS. This device type allows sending standard SDO's to a generic DS406 based module.

In order the DS406 device via the EAS, the Device Functionality should be of type General Device and the Device Type should be DS406 as per illustration below:

The user may also choose whether to support the DS406 heartbeat mechanism (GMAS as consumer). This can be set by selecting the Heartbeat Support options.

Calling DS406 devices from within the code is identical to any motion related device. An *AxisRef* must be obtained in the identical way.

Only some of the functions support DS406.

The resource file has a new cluster by the name of <GENERAL_DEVICES>. This cluster also holds DS406 related information. For instance, if I were to define a DS406 device that supports the DS301 heartbeat mechanism, it will look as follows:

```
<GENERAL_DEVICES>

        <NODE NAME="a01"  TYPE="ds406" ID="1" HBT_SUPPORT="1"/>

</GENERAL_DEVICES>
```

4.10 New Data Recording Features

- Additional trigger option (On Begin) was added to the trigger list.
- Additional parameters can now be recorded:
    o Desired Velocity.
    o Motor Current
    o UI and UF from / to drive. This is reflected in the user integer/float parameters. These parameters may be accesses via the Read/Write parameters API.

4.11 Motion Abortion as per PLCopen definitions

In the previous GMAS version, the abortion flag in the function block did not work

according to the PLCopen, and the motion aborted to speed 0. In this version, the

abortion is according to PLCopen.

4.12 Ability to send SDO's from user programs and from RPC host

A new API was added by the name of SendSDO(). This function is currently synchronous (in the library level). Meaning – the function does not return until a reply is received.

4.13 USB support for Basic System Configuration

This version supports a USB interface for configuring the GMAS in order to be able to work via the standard Ethernet port.

The following functions are supported:

- Change / Read IP Address.

- Change / Read Gateway.

- Change / Read Subnet mask.

- Change / Read Server IP.

- Change / Read download version path.

- Change / Read DHCP

- Change / Read GMAS name

Please refer to the GMAS – USB Command Reference for further information.

4.14 New U-Boot supporting enhanced version update mechanism including auxiliary backup image

The new GMAS firmware and u-boot version (from version 8), support a feature where the GMAS always holds 2 images. The current image, and a previous image. In case of download firmware error that caused the FLASH to be corrupt, the previous image will be loaded.

4.15 Enhanced Events Mechanism

The GMAS supports event mechanism where a callback function is called due to an

event that occurred in the GMAS / Drive. These events must be registered. The event

mechanism is possible from the local 'C' programming and from the remote (remote

computer) programming interfaces.

All the user needs to do is the following:

- Register the Callback function within the MMC_InitConnection command.
- Register the type of events to be called and call the MC_OpenUdpChannelCmd command.
- Create the callback function.

The following callback events are supported:

#define EMCY_EVT   5   - Due to emergency received by GMAS

#define MOTIONENDED_EVT   6   - Specific Drive finished motion

#define HBEAT_EVT   7   - Heartbeat error on a specific drive

#define PDORCV_EVT   8   - PDO Receive error

#define DRVERROR_EVT   9   - Drive Error

#define EMIT_EVT   10   - Emit. Not yet supported.

#define HOME_ENDED_EVT   11   - Homing Ended on a specific drive.

#define SYSTEMERROR_EVT   12   - System Error.

#define MODBUS_WRITE_EVT 13   - Modbus written

For example:

Register the Callback function within the *MMC_InitConnection* command:

```
int (*fun_ptr)(unsigned char*, short,LPSOCKADDR);

fun_ptr = MyCallbackFunc ;

//

int rc = MMC_InitConnection(MMC_RPC_CONN_TYPE, stConnect, (MMC_CB_FUNC)fun_ptr,

&conn_hndl);
```

Register the type of events to be called and call the *MMC_OpenUdpChannelCmd* command:

```
open_udp_in.iPort = 5000;

open_udp_in.iEventsMask = 0xefffffff;        // ALL Events Mask

open_udp_in.cFirst = 10;

open_udp_in.cSecond = 10;

open_udp_in.cThird = 20;

open_udp_in.cFourth = 48;

//

MMC_OpenUdpChannelCmd(conn_hndl,&open_udp_in,&open_udp_out) ;
```

Create the callback function:

```
int MyCallbackFunc(unsigned char* rcvBuff, short rcvBuffSize,void*
lpsock)
{
```

```c
        printf("size: %d ", rcvBuffSize) ;

//

// Which function ID was received ...

switch(rcvBuff [1])

{

case EMCY_EVT:

        printf("Emergency Event received\r\n") ;

        break ;

case MOTIONENDED_EVT:

        printf("Motion Ended Event received\r\n") ;

        break;

case HBEAT_EVT:

        printf("H Beat Fail Event received\r\n") ;

        break;

case PDORCV_EVT:

        printf("PDO Received Event received\r\n") ;

        break;

case DRVERROR_EVT:

        printf("Drive Error Received Event received\r\n") ;

        break;

case HOME_ENDED_EVT:

        printf("Home Ended Event received\r\n") ;

        break;

case SYSTEMERROR_EVT:

        printf("System Error Event received\r\n") ;

        break;

case MODBUS_WRITE_EVT:

        printf("Modbus Write Event received\r\n") ;

        break;

}
```

```
        //

        return 1 ;

    }
```

The above events are GMAS related. For instance – if an emergency on ANY of the drives is

received by the GMAS, then the event is triggered. This is not the case for *Motion Ended* and

DS401 events.

In order to receive *motion ended* events and *DS401 events*, the user must call specific

Functions:

- MMC_EnableDS401DIChangedEvent
- MMC_EnableMotionEndedEventCmd

4.16 Function attributes stating the function call context

API functions were given attributes. These attributes, which can be enhanced in the future, enable a quick check regarding whether the function can be called as per the *AxisRef* that is sent to the function. For instance – the *MoveAbsolute* function can not be called for a DS401 *AxisRef*. This mechanism allows quick error checking on function calls, and reduces the chance to 'forget' to insert protections. Calling a function that does not exist, for a specific node was unexpected.

4.17 New resource file definitions

In order to support all of the above new functions, the resource file went through some modifications. The following is an example to a resource file with the following credentials:

Global Parameters:

- CAN network.
- 3 ms SYNC
- 1Mb baud.

Motion Parameters:

- 3 axes
- All axes NC
- No HBeat.

IO Devices

- 1 IO device.
- Input and output,
- No HBeat support.

Group

- 2 groups
- Etc …

```xml
<?xml version="1.0"?>

<RESOURCES NAME="MMCResources" TYPE="GMAS">

        <GLOBAL_PARAMS>

                <NAME  CYCLE_TIME="3000"/>

                <NAME  NET_TYPE="CAN"/>

                <NAME  BAUDRATE="1000"/>

                <NAME  ENQUIRE_FB_STATUS="0"/>

                <NAME  RES_ID="1025"/>

        </GLOBAL_PARAMS>

        <MOTION_DEVICES>

                <NODE NAME="a01" TYPE="ds402" SUBTYPE="ELMO" ID="1" NC_AXIS="0" MODULO="0" DIVIDER="0"
DRIVE_TYPE="SimpleIQ" DOUT_SUPPORT="0" HBT_SUPPORT="0"/>

                <NODE NAME="a02" TYPE="ds402" SUBTYPE="ELMO" ID="2" NC_AXIS="0" MODULO="0" DIVIDER="0"
DRIVE_TYPE="SimpleIQ" DOUT_SUPPORT="0" HBT_SUPPORT="0"/>

                <NODE NAME="a03" TYPE="ds402" SUBTYPE="ELMO" ID="3" NC_AXIS="0" MODULO="0" DIVIDER="0"
DRIVE_TYPE="SimpleIQ" DOUT_SUPPORT="0" HBT_SUPPORT="0"/>

        </MOTION_DEVICES>


        <IO_DEVICES>

                <NODE NAME="i01"  TYPE="ds401" ID="4" IO_MODULE_TYPE="3"/>

        </IO_DEVICES>DEVICES>

        <GENERAL_DEVICES>

        </GENERAL_DEVICES>

        <VIRTUAL_OBJECTS>

                <VIRTUAL_OBJECT NAME="v01" TYPE="VECTOR" ID="10" DIM="3" MODULO="0" DIVIDER="0">
```

```
            <MEMBER NAME="a01" INDEX_IN_GROUP="0"/>

            <MEMBER NAME="a02" INDEX_IN_GROUP="1"/>

            <MEMBER NAME="a03" INDEX_IN_GROUP="2"/>

        </VIRTUAL_OBJECT>

        <VIRTUAL_OBJECT NAME="v02" TYPE="VECTOR" ID="10" DIM="2" MODULO="0" DIVIDER="0">

            <MEMBER NAME="a01" INDEX_IN_GROUP="0"/>

            <MEMBER NAME="a02" INDEX_IN_GROUP="1"/>

        </VIRTUAL_OBJECT>


    </VIRTUAL_OBJECTS>

</RESOURCES>
```

4.18 New library APIs and interfaces to the GMAS in order to support all of the above

SetGMASName()

GetGMASName()

ReadAxisError()

SendSdo()

All of the following DS401 functions:

MMC_ReadDS401DIGroup

MMC_WriteDS401DOGroup

MMC_EnableDS401DIChangedEvent

MMC_DisableDS401DIChangedEvent

MMC_ReadDS401DInput

MMC_WriteDS401DOutput

4.19 Bug Fixes

1. AxisRef/AxisHandle bug – did not correlate to the handle in the resource file.
2. Kinematic transformations bug fix. Used to support only 1:1 ratios.
3. Overshoots – many Profiler overshoot bug fixes.
4. FB management – reentrancy and synchronization problems User - Kernel.
5. mmc_reset bug fix.
6. Reactor – Can Dispatcher Thread – context switch bug fix.

7. Reactor – Can Dispatcher Thread – synchronization bug fix.

8. Can driver synchronization bug – HPT versus user fix.

9. GetVersion API modification – removing the read from FLASH, the read is now from global parameter.

10. Motor On/Off FSTM modification.

11. Mounting of JFFS partition is now performed after watchdog feeder daemon invocation due to watchdog errors.

12. Locator – extending the protocol.

13. Download Version – adding a CRC check on the actual received image bug fix.

14. NetworkInfo() API – bug fix.

15. GMAS as a gateway bug fix, NC/Non NC.

16. Uboot - Image Internal CRC validation.

17. PDO#3/PDO#4 mapped as RPDO.

18. WinSCP - /dev/null permission bug fix.

ॐ౫౦