Gold Line

G-MAS Administrative and Motion API

Administrative and Motion API for the Gold Maestro Network Motion Controller

February 2013 (Ver. 1.501)

Elmo Motion Control

www.elmomc.com

Important Notice

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

- This document is copyrighted and all rights are reserved by Elmo Motion Control Ltd. This product may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, by Elmo Motion Control Ltd.
- This document contains proprietary information belonging to Elmo Motion Control Ltd. Such information is supplied solely for assisting users of Gold Maestro Network Motion Controller.
- The text and graphics included in this document are for the purpose of illustration and reference only. The specifications on which they are based are subject to change without notice.
- Elmo Motion Control and the Elmo Motion Control logo are trademarks of Elmo Motion Control Ltd.

Document no. MAN-G-MAS-API Ver 1.501

Copyright © 2013
Elmo Motion Control Ltd.
All rights reserved.

Revision History

Version	Release Date	Changes/Remarks
0.00	October 2010	Preliminary Draft
0.01	29th Dec 2010	Pre-Release Draft containing additional, single axis administrative, multiple Axes, CANbus, and EtherCAT, function blocks.
0.02	26th Jan 2011	Updated functions added and corrected.
0.03	13th Feb 2011	All structures in code updated to standard Elmo format.
1.0	1st March 2011	Release version
1.1	Apr 2012	Format changed and document edited. New functions added.
1.2	Sept 2012	G-MAS Hardware/Software Limits Handling and new function and PVT Motion to include five new functions. API Events updated.
1.3	Nov 2012	Updated functions for G-MAS Version 1.1.1.5. Addition of errors, explanations, and corrections to functions. Addition of G-MAS PC Configuration for static IP address
1.400	Dec 2012	Additional Events callbacks, and EthernetIP for C++

Version	Release Date	Changes/Remarks
1.500	Jan 2013	Addition of move Polynomial function block, Administrative Function Block Support, Super imposed motions, VxWorks library interfaces, Additional Transition type for Blended ACS motion, CAN configuration interfaces enabling PDO ,SDO configuration via resource file, TCP/IP and UDP, CPP libraries. Addition of SEND_ASYNC_EVENT event mask. Corrected Recording Triggers.
1.501	Feb 2013	Addition of errors

Elmo Worldwide

Head Office

Elmo Motion Control Ltd.

64 Gisin St., P.O. Box 463, Petach Tikva 49103
Israel

Tel: +972 (3) 929-2300 • Fax: +972 (3) 929-2322 • info-il@elmomc.com

North America

Elmo Motion Control Inc.

42 Technology Way, Nashua, NH 03060
USA

Tel: +1 (603) 821-9979 • Fax: +1 (603) 821-9943 • info-us@elmomc.com

Europe

Elmo Motion Control GmbH

Steinkirchring 1, D-78056, Villingen-Schwenningen
Germany

Tel: +49 (0) 7720-85 77 60 • Fax: +49 (0) 7720-85 77 70 • info-de@elmomc.com

China

Elmo Motion Control Technology (Shanghai) Co. Ltd.

Room 1414, Huawen Plaza, No. 999 Zhongshan West Road, Shanghai (200051)
China

Tel: +86-21-32516651 • Fax: +86-21-32516652 • info-asia@elmomc.com

Asia Pacific

Elmo Motion Control APAC Ltd.

B-601 Pangyo Innovalley, 621 Sampyeong-dong, Bundang-gu, Seongnam-si, Gyeonggi-do,
South Korea (463-400)

Tel: +82-31-698-2010 • Fax: +82-31-8018-078 • info-asia@elmomc.com

Table of Contents

Chapter 1: Introduction 1-16

1.1. G-MAS over (Go) Standard 1-16
1.2. What the API Does 1-17
1.3. Terminology 1-18
1.4. How to Use this Document 1-22

Chapter 2: G-MAS Overview 2-26

2.1. For EAS 2-30
2.2. G-MAS Operation Modes 2-31
2.2.1. NC Motions 2-31
2.2.2. Distributed / Standard DS-402 (stand-alone) Drive 2-31
2.2.3. G-MAS Axes and Node Definitions 2-31
2.2.4. G-MAS to Servo Drive Interfaces 2-33

Chapter 3: G-MAS Hardware and Software Limits Handling 3-34

3.1. Introduction 3-34
3.2. Interfaces 3-35
3.2.1. Software Position Limits 3-35
3.2.2. Status Register 3-36
3.2.3. ACS\SingleAxis 3-37
3.2.4. MCS 3-37
3.2.5. MCS Limit Register (32 bits) 3-37
3.2.6. Stop Parameters 3-38
3.2.7. Stop-on-Limit Event 3-38
3.3. Function Block Pre-Insertion Behavior 3-39
3.3.1. Single Axis 3-39
3.3.2. Multi Axes Group 3-40
3.4. Real Time Behavior 3-41
3.5. System Constraints And Limitations 3-42

Chapter 4: G-MAS Initial Connection 4-44

4.1. G-MAS USB Connection 4-44
4.1.1. Connection Procedure 4-44
4.1.2. dhcp – Static / Dynamic IP 4-45
4.1.3. ipaddr – GMAS IP Address 4-46
4.1.4. ipmask – GMAS IP Subnet Mask 4-46
4.1.5. defgateway – GMAS Default Gateway 4-46
4.2. G-MAS PC configuration 4-48
4.2.1. XP Windows Setup 4-48
4.2.2. Windows 7 Seup 4-53

Chapter 5: Motion and Administrative Function Blocks 5-59
5.1. Compliance and Portability 5-59
5.2. Function Block States 5-59
5.2.1. Single Axis 5-59
5.2.2. Group of Axes 5-61
5.2.3. Function Block Status Bit Masks 5-62
5.3. Axis, Group, Global, Parameters 5-64
5.3.1. Legend 5-64
5.3.2. Parameters Tables 5-65
5.4. Axis Status 5-69
5.4.1. Blended Behaviour Mechanism 5-71
5.4.2. Special function block insertion mechanism 5-72
5.5. Administrative Function Block Handling 5-73
5.5.1. Classification 5-73
5.5.2. Flags 5-74
5.5.3. Examples 5-74
5.6. Single Axis Motion Control 5-76
5.6.1. MMC_Halt 5-77
5.6.2. MMC_Home 5-81
5.6.3. MMC_HomeDS402 5-96
5.6.4. MMC_MoveAbsolute 5-101
5.6.5. MMC_MoveAdditive 5-107
5.6.6. MMC_MoveRelative 5-113
5.6.7. MMC_MoveVelocity 5-119
5.6.8. MMC_MoveAbsoluteRepetitive 5-125
5.6.9. MMC_MoveRelativeRepetitive 5-130
5.6.10. MMC_MoveAdditiveRepetitive 5-135
5.6.11. MMC_Stop 5-140
5.7. Single Axis Administrative Control 5-145
5.7.1. Elmo SuperImposed Motion 5-146
5.7.2. Special Function 5-147
5.7.3. MMC_GetFBDepth 5-148
5.7.4. MMC_GetTotalFbDepth 5-151
5.7.5. MMC_Power 5-154
5.7.6. MMC_PositionProfile 5-158
5.7.7. MMC_ReadActualPosition 5-161
5.7.8. MMC_ReadActualTorque 5-164
5.7.9. MMC_ReadActualVelocity 5-167
5.7.10. MMC_ReadAxisError 5-170
5.7.11. MMC_ReadBoolParameter 5-173
5.7.12. MMC_GlobalReadBoolParameter 5-176
5.7.13. MMC_ReadDigitalInput(s) 5-179
5.7.14. MMC_ReadDigitalOutputs 5-184
5.7.15. MMC_ReadDigitalOutputs32Bit 5-187

5.7.16. MMC_ReadParameter 5-190
5.7.17. MMC_GlobalReadParameter 5-193
5.7.18. MMC_ReadStatus 5-196
5.7.19. MMC_Reset 5-199
5.7.20. MMC_SetOverride 5-202
5.7.21. MMC_SetPosition 5-206
5.7.22. MMC_TouchProbeEnable 5-209
5.7.23. MMC_TouchProbeDisable 5-212
5.7.24. MMC_WriteBoolParameter 5-215
5.7.25. MMC_GlobalWriteBoolParameter 5-218
5.7.26. MMC_WriteDigitalOutputs 5-221
5.7.27. MMC_WriteDigitalOutputs32Bit 5-224
5.7.28. MMC_WriteParameter 5-227
5.7.29. MMC_GlobalWriteParameter 5-230
5.7.30. MMC_AxisLink 5-233
5.7.31. MMC_AxisUnLink 5-236
5.8. Multiple Axes Motion Control 5-238
5.8.1. Coordinate System and kinematic transformation 5-238
5.8.2. Move Polynomial Function Block 5-264
5.8.3. Splines 5-267
5.8.4. NC_MCS_Info_Struct 5-273
5.8.5. NC_MCS_Kin_Ref_Struct 5-274
5.8.6. MMC_GroupStop 5-275
5.8.7. MMC_GroupHalt 5-279
5.8.8. MMC_MoveCircularAbsolute 5-283
5.8.9. MMC_MoveCircularAbsoluteCenter 5-291
5.8.10. MMC_MoveCircularAbsoluteBorder 5-297
5.8.11. MMC_MoveCircularAbsoluteRadius 5-303
5.8.12. MMC_MoveCircularAbsoluteAngle 5-310
5.8.13. MMC_MoveLinearAbsolute 5-315
5.8.14. MMC_MoveLinearRelative 5-322
5.8.15. MMC_MoveLinearAdditive 5-329
5.8.16. MMC_MoveLinearAdditiveEx 5-334
5.8.17. MMC_MoveLinearAbsoluteRepetitive 5-339
5.8.18. MMC_MoveLinearRelativeRepetitive 5-344
5.8.19. MMC_PathSelect 5-349
5.8.20. MMC_MovePath 5-353
5.8.21. MMC_PathUnselect 5-358
5.9. Multiple Axes Administrative Control 5-361
5.9.1. MMC_AddAxisToGroup 5-362
5.9.2. MMC_GroupDisable 5-365
5.9.3. MMC_GroupEnable 5-368
5.9.4. MMC_GroupReadActualPosition 5-371
5.9.5. MMC_GroupReadActualVelocity 5-374

5.9.6. MMC_GroupReadError 5-377
5.9.7. MMC_GroupReadStatus 5-380
5.9.8. MMC_GroupReset 5-383
5.9.9. MMC_GroupSetOverride 5-386
5.9.10. MMC_GroupSetPosition 5-391
5.9.11. MMC_RemoveAxisFromGroup 5-395
5.9.12. MMC_SetKinTransform 5-398
5.9.13. MMC_GroupReadParameter 5-403
5.9.14. MMC_GroupReadBoolParameter 5-406
5.9.15. MMC_GroupWriteParameter 5-409
5.9.16. MMC_GroupWriteBoolParameter 5-412

Chapter 6: Error Handling 6-415

6.1. Buffered Errors 6-416
6.2. G-MAS Error IDs 6-417
6.2.1. Error correction error IDs 6-433
6.3. Continued G-MAS Error IDs 6-434
6.4. G-MAS PVT Motion Error IDs 6-443
6.5. Continued G-MAS Error IDs 6-444
6.6. NC Driver Warning IDs 6-452
6.7. NC Profiler Error IDs 6-453
6.8. NC Profiler Caution IDs 6-461
6.9. Internal Library Errors 6-463
6.10. EtherNetIP Communication Error Codes 6-465

Chapter 7: Error Correction Mechanism 7-467

7.1. 2-D Error Correction 7-467
7.2. 3-D Error Correction 7-469
7.3. Data Representation 7-470
7.3.1. 1-D Representation 7-470
7.3.2. 2-D Representation 7-470
7.3.3. 3-D Representation 7-471
7.4. Error Correction Functions 7-472
7.4.1. MMC_LoadErrorCorrTable 7-473
7.4.2. MMC_EnableErrorCorrTable 7-476
7.4.3. MMC_GetErrorTableStatus 7-479
7.4.4. MMC_DisableErrorCorrTable 7-482
7.4.5. MMC_UnloadErrorCorrTable 7-485

Chapter 8: Bulk Parameters Reading 8-488

8.1. Bulk Reading Functions 8-488
8.1.1. MMC_ConfigBulkRead 8-489
8.1.2. MMC_PerformBulkRead 8-497

Chapter 9: Position, Velocity, Time (PVT) Motion 9-504

9.1. Overview ... 9-504
9.2. PV, PVT Profiler ... 9-504
9.3. Data loading ... 9-504
9.4. PVT motion ... 9-505
9.5. On-The-Fly Mode ... 9-505
9.5.1. Initializing Table ... 9-505
9.5.2. Loading Data ... 9-505
9.5.3. Cyclic Mode ... 9-506
9.6. File example ... 9-506
9.7. PVT Functions ... 9-507
9.7.1. MMC_InitTable ... 9-508
9.7.2. MMC_LoadTableFromFile ... 9-512
9.7.3. MMC_UnloadTable ... 9-515
9.7.4. MMC_MoveTable ... 9-518
9.7.5. MMC_AppendPointsToTable ... 9-522
9.7.6. MMC_GetTableIndex ... 9-526

Chapter 10: API Services and Operations ... 10-530

10.1. Download Firmware and Errors ... 10-532
10.2. Type Definitions ... 10-533
10.2.1. MMC_DOWNLOAD_TYPE_ENUM ... 10-533
10.2.2. MMC_CONNECT_HNDL ... 10-533
10.3. Main Configuration Function Blocks ... 10-534
10.3.1. MMC_ChangeToPreOPMode ... 10-535
10.3.2. MMC_ChangeToOperationMode ... 10-538
10.3.3. MMC_ClearNodeFbList ... 10-541
10.3.4. MMC_CmdStatus ... 10-544
10.3.5. MMC_CloseConnection ... 10-547
10.3.6. MMC_Config ... 10-549
10.3.7. MMC_CreateSYNCTimer ... 10-552
10.3.8. MMC_DestroySYNCTimer ... 10-554
10.3.9. MMC_DownloadFoE ... 10-555
10.3.10. MMC_Dwell ... 10-560
10.3.11. MMC_Exit ... 10-563
10.3.12. MMC_FreeFbStat ... 10-566
10.3.13. MMC_GetActiveVectorsNum ... 10-569
10.3.14. MMC_GetErrorCodeDescriptionByID ... 10-572
10.3.15. MMC_GetFoEStatus ... 10-575
10.3.16. MMC_GetEnquireFbStatus ... 10-581
10.3.17. MMC_GetAxisByName ... 10-584
10.3.18. MMC_GetGroupByName ... 10-587
10.3.19. MMC_GetGroupMembersInfo ... 10-590
10.3.20. MMC_GetGMASOperationMode ... 10-595
10.3.21. MMC_GetStatusRegister ... 10-598

10.3.22. MMC_GetResList 10-601
10.3.23. MMC_GetResSnapshot 10-604
10.3.24. MMC_GetVersion 10-607
10.3.25. MMC_GetVersionEx 10-610
10.3.26. MMC_InitConnection 10-613
10.3.27. MMC_IPCInitConnection 10-615
10.3.28. MMC_LoadParam 10-617
10.3.29. MMC_RpcInitConnection 10-620
10.3.30. MMC_RpcInitConnectionEx 10-622
10.3.31. MMC_ResetMultiAxisControl 10-624
10.3.32. MMC_ResExportFile 10-627
10.3.33. MMC_ResImportFile 10-630
10.3.34. MMC_SaveParam 10-633
10.3.35. MMC_SetEnquireFbStatus 10-636
10.3.36. MMC_SetDefaultParameters 10-638
10.3.37. MMC_SetDefaultParametersGlobal 10-641
10.3.38. MMC_SetIsToLoadGlobalParams 10-643
10.3.39. MMC_ShowNodeStat 10-645
10.3.40. MMC_GetActiveAxesNum 10-648
10.3.41. MMC_ToggleConsoleOutput 10-651
10.3.42. MMC_GetCyclesCounter 10-653
10.3.43. MMC_WriteGroupOfParameters 10-656
10.3.44. MMC_ReadGroupOfParameters 10-661
10.3.45. MMC_GetVerPath 10-664
10.3.46. MMC_DownloadVersion 10-664
10.3.47. MMC_ReadDownloadVersionStatus 10-664
10.3.48. MMC_SetVerPath 10-664

Chapter 11: Connectivity and Configuration 11-665

11.1. Network Function Blocks 11-665
11.1.1. MMC_CloseUdpChannel 11-666
11.1.2. MMC_GetDefGateway 11-669
11.1.3. MMC_GetDhcp 11-672
11.1.4. MMC_GetIpAddr 11-675
11.1.5. MMC_GetIpMask 11-678
11.1.6. MMC_GetServerIp 11-681
11.1.7. MMC_NetworkInfo 11-684
11.1.8. MMC_NetworkScan 11-690
11.1.9. MMC_OpenUdpChannel 11-693
11.1.10. MMC_SetDefGateway 11-696
11.1.11. MMC_SetDhcp 11-699
11.1.12. MMC_SetIpAddr 11-702
11.1.13. MMC_SetIpMask 11-705
11.1.14. MMC_SetServerIp 11-708

11.2. Host Communication 11-711
11.3. Modbus Communication Function Blocks 11-711
11.3.1. MMC_MbusIsRunning 11-712
11.3.2. MMC_MbusReadCoilsTable 11-715
11.3.3. MMC_MbusReadHoldingRegisterTable 11-718
11.3.4. MMC_MbusReadInputsTable 11-721
11.3.5. MMC_MbusStartServer 11-724
11.3.6. MMC_MbusStopServer 11-727
11.3.7. MMC_MbusWriteCoilsTable 11-730
11.3.8. MMC_MbusWriteHoldingRegisterTable 11-733
11.4. CANbus Drive Communication 11-736
11.4.1. Master – Slave Relations 11-737
11.4.2. CANopen DS-402 Modes of Operation 11-737
11.4.3. PDO Mapping 11-738
11.4.4. Using Event Groups 16 and 17 11-740
11.4.5. Servo Drive Sub-Index 11-741
11.4.6. SYNC and Time Stamp 11-741
11.4.7. CAN Bulk Upload 11-742
11.4.8. CAN – PDO, SDO Configurator 11-742
11.5. CANbus Function Blocks 11-743
11.5.1. MMC_CancelVirtualEncoder 11-744
11.5.2. MMC_CancelParamEvPDO3 11-746
11.5.3. MMC_CancelParamEvPDO4 11-749
11.5.4. MMC_CfgRegParamEvPDO3 11-752
11.5.5. MMC_CfgRegParamEvPDO4 11-756
11.5.6. MMC_CfgUserParamEvPDO3 11-760
11.5.7. MMC_CfgUserParamEvPDO4 11-764
11.5.8. MMC_ChangeDefaultPDOConfiguration 11-769
11.5.9. MMC_ChngOpMode 11-772
11.5.10. MMC_ConfigEventModePDO3 11-775
11.5.11. MMC_ConfigEventModePDO4 11-778
11.5.12. MMC_ConfigVirtualEncoder 11-781
11.5.13. MMC_GetAxisByCanId 11-784
11.5.14. MMC_GetPDOInfo 11-787
11.5.15. MMC_GetSyncTime 11-790
11.5.16. MMC_PDOGeneralRead 11-793
11.5.17. MMC_PDOGeneralWrite 11-796
11.5.18. MMC_ReceiveCANRawData 11-799
11.5.19. MMC_SendCANRawData 11-802
11.5.20. MMC_SendCmd 11-805
11.5.21. MMC_SetHeartBeatConsumer 11-808
11.5.22. MMC_SetSyncTime 11-811
11.5.23. MMC_StartBulkUpload 11-814
11.5.24. MMC_GetBulkUploadStatus 11-817

11.5.25. MMC_GetBulkUploadData 11-820
11.6. EtherCAT Drive Communication 11-823
11.6.1. Elmo EtherCAT 11-824
11.6.2. Elmo Slave Drives 11-825
11.6.3. EtherCAT with G-MAS 11-826
11.6.4. EtherCAT Gateway 11-827
11.7. EtherCAT and CANbus Function Blocks 11-827
11.7.1. MMC_DisableEthercatConfigMode 11-828
11.7.2. MMC_EnableEthercatConfigMode 11-831
11.7.3. MMC_ECATIODisableDIChangedEvent 11-833
11.7.4. MMC_ECATIOEnableDIChangedEvent 11-836
11.7.5. MMC_ECATIOReadDigitalInput 11-839
11.7.6. MMC_ECATIOReadAnalogInput 11-842
11.7.7. MMC_ECATIOWriteAnalogOutput 11-845
11.7.8. MMC_ECATIOWriteDigitalOutput 11-848
11.7.9. MMC_GetCommStatistics 11-851
11.7.10. MMC_GetEthercatCommStatistics 11-855
11.7.11. MMC_GetCommDiagnostics 11-864
11.7.12. MMC_GetReactorStatistics 11-868
11.7.13. MMC_IsEthercatConfigMode 11-871
11.7.14. MMC_ResetCommDiagnostics 11-874
11.7.15. MMC_ResetCommStatistics 11-877
11.7.16. MMC_SendSDO 11-880
11.8. Interpreter Command Functions 11-884
11.8.1. Get Function – Asynchronous Mode 11-884
11.8.2. MMC_ElmoExecuteLabel 11-886
11.8.3. MMC_ElmoSetParameter 11-889
11.8.4. MMC_ElmoGetParameter 11-891
11.8.5. MMC_ElmoGetArray 11-893
11.8.6. MMC_ElmoGetArrayAndRetrieveData 11-895
11.8.7. MMC_ElmoGetParameterAndRetrieveData 11-897
11.8.8. MMC_ElmoSetArray 11-899
11.8.9. MMC_ElmoQueryOperationFIFOIndex 11-901
11.8.10. MMC_ElmoQueryOperationFIFORetrieveData 11-902
11.8.11. MMC_ElmoQueryOperationFIFOIndexReset 11-904
11.8.12. MMC_ElmoCall 11-905
11.9. EtherNetIP Communication 11-906
11.9.1. Terminology 11-906
11.9.2. Configuring the Ethernet IP Device as Adapter 11-907
11.9.3. Ethernet/IP Setup 11-913
11.10. EtherNetIP Functions 11-915
11.10.1. EipGetAdpTagRefByName 11-916
11.10.2. EipWriteAdpTag 11-918
11.10.3. EipReadAdpTag 11-921

11.10.4. EipGetAssemblyRefByInstance 11-923
11.10.5. EipGetAssemblyRefByName 11-925
11.10.6. EipSetAssembly 11-927
11.10.7. EipGetAssembly 11-929
11.10.8. EipGetDevTagRefByName 11-931
11.10.9. EipSetDevTag 11-933
11.10.10. EipGetDevTag 11-935
11.10.11. EipReadDevTagData 11-937
11.10.12. EipSyncGetDevTag 11-939
11.10.13. EipCheckDevTagReply 11-941
11.10.14. EipOpenSession 11-943
11.10.15. EIPCloseSession 11-945
11.10.16. EipCreate 11-947
11.10.17. EipDestroy 11-949
11.10.18. Functions and Implementation Example 11-951
11.11. DS-401 Drive Communication 11-956
11.12. DS-401 Function Blocks 11-956
11.12.1. MMC_CancelGeneralRPDO3 11-957
11.12.2. MMC_CancelGeneralRPDO4 11-960
11.12.3. MMC_CancelGeneralTPDO3 11-963
11.12.4. MMC_CancelGeneralTPDO4 11-966
11.12.5. MMC_ConfigGeneralRPDO3 11-969
11.12.6. MMC_ConfigGeneralRPDO4 11-972
11.12.7. MMC_ConfigGeneralTPDO3 11-975
11.12.8. MMC_ConfigGeneralTPDO4 11-978
11.12.9. MMC_DisableDS401DIChangedEvent 11-981
11.12.10. MMC_EnableDS401DIChangedEvent 11-984
11.12.11. MMC_ReadDS401DIGroup 11-987
11.12.12. MMC_ReadDS401DInput 11-990
11.12.13. MMC_WriteDS401DOGroup 11-993
11.12.14. MMC_WriteDS401DOutput 11-996

Chapter 12: API Events 12-999

12.1. Communication Byte Order 12-1000
12.2. Communication ASYNC Replies (Events) From Drives 12-1000
12.3. Download Firmware Notifications 12-1001
12.4. Emergency Event 12-1002
12.5. Motion Ended Event 12-1002
12.6. Heart Beat Event 12-1003
12.7. PDO Receive Event 12-1003
12.7.1. Event Group equals to 5 or 6 12-1004
12.7.2. Event Group equals to 11 12-1004
12.7.3. Event Group Equals to 16 or 17 12-1004
12.7.4. Event Group equals to 1 – 15 besides 5, 6, 11, 16 and 17 12-1005

12.8. Home Ended Event 12-1005
12.9. Modbus Write Event 12-1005
12.10. Touch Probe Ended Event 12-1006
12.11. Node Connected Event 12-1006
12.12. Node Error Event 12-1007
12.13. Stop ON Limit Event 12-1007
12.14. Table Underflow Event 12-1007
12.15. Global Async Reply Event 12-1008
12.16. Ethernet-IP Event 12-1008
12.17. Communication Event Mechanism 12-1009
12.18. Events Mask and Enumeration 12-1010
12.19. Asynchronous Events Callback 12-1011
12.19.1. Callback Prototype 12-1011
12.19.2. Data Structure 12-1011
12.19.3. Event Extraction Example 12-1011
12.19.4. Net To local Conversion 12-1016
12.20. Events Function Blocks 12-1018
12.20.1. MMC_ClearEventsMask 12-1019
12.20.2. MMC_DisableMotionEndedEvent 12-1022
12.20.3. MMC_EnableMotionEndedEvent 12-1025
12.20.4. MMC_GetEventsMask 12-1028
12.20.5. MMC_SetEventsMask 12-1031

Chapter 13: G-MAS Personality 13-1034

13.1. G-MAS Personality Functions 13-1035
13.1.1. MMC_CreatePersonalityParameters 13-1036

Chapter 14: Data Recording 14-1038

14.1. Triggering a Recording 14-1038
14.2. Active Range Support 14-1039
14.3. Using Data Recording in the G-MAS 14-1040
14.3.1. Excluding Triggers 14-1040
14.3.2. Including Triggers 14-1040
14.4. Recording Definitions and Parameters 14-1042
14.4.1. Recording Data Signals Bitmask Definitions 14-1042
14.4.2. Recording Signal Parameters 14-1042
14.4.3. Trigger Modes 14-1047
14.5. Data Recording Functions 14-1050
14.5.1. MMC_BeginRecording 14-1050
14.5.2. MMC_StopRecording 14-1054
14.5.3. MMC_UploadData 14-1057
14.5.4. MMC_RecStatus 14-1060
14.5.5. MMC_UploadDataHeader 14-1063

Chapter 15: Using Function Blocks in C++ 15-1068

15.1. Programming in C++ 15-1068
15.1.1. CMMCException 15-1069
15.2. The CMMCAxis class 15-1070
15.2.1. CMMCDS406 15-1070
15.2.2. DisableMotionEndedEvent 15-1071
15.2.3. EnableMotionEndedEvent 15-1072
15.2.4. SetDefaultManufacturerParameters 15-1072
15.2.5. GetFbDepth 15-1073
15.2.6. GetAxisByName 15-1073
15.2.7. GetGroupAxisByName 15-1074
15.2.8. SetBoolParameter 15-1075
15.2.9. SetParameter 15-1076
15.2.10. GetBoolParameter 15-1077
15.2.11. GetParameter 15-1078
15.3. The CMMCSingleAxis class 15-1079
15.3.1. SetDefaultParams 15-1082
15.3.2. SetDefaultHomeDS402Params 15-1083
15.3.3. SetDefaultHomeParams 15-1084
15.3.4. Home 15-1085
15.3.5. HomeDS402 15-1086
15.3.6. MoveAbsolute 15-1087
15.3.7. MoveAdditive 15-1088
15.3.8. MoveRelative 15-1089
15.3.9. MoveVelocity 15-1090
15.3.10. MoveAbsoluteRepetitive 15-1091
15.3.11. MoveRelativeRepetitive 15-1092
15.3.12. MoveAdditiveRepetitive 15-1093
15.3.13. PositionProfile 15-1094
15.3.14. TouchProbeDisable 15-1095
15.3.15. TouchProbeEnable 15-1095
15.3.16. SetOpMode 15-1096
15.3.17. GetOpMode 15-1096
15.3.18. PowerOn 15-1097
15.3.19. PowerOff 15-1097
15.3.20. GetActualPosition 15-1098
15.3.21. GetActualVelocity 15-1098
15.3.22. GetActualTorque 15-1099
15.3.23. Halt 15-1099
15.3.24. Stop 15-1100
15.3.25. GetAxisError 15-1101
15.3.26. GetDigInput[s] 15-1101
15.3.27. GetDigOutputs32Bit 15-1102
15.3.28. GetDigOutputs 15-1102
15.3.29. SetDigOutputs32Bit 15-1103

15.3.30. SetDigOutputs 15-1104
15.3.31. SetOverride 15-1105
15.3.32. ConfigPDO 15-1106
15.3.33. CancelPDO 15-1109
15.3.34. ConfigPDOEventMode 15-1109
15.3.35. ChangeDefaultPDOConfig 15-1110
15.3.36. ElmoSetAsyncParam 15-1111
15.3.37. ElmoGetAsyncIntParam 15-1112
15.3.38. ElmoGetAsyncFloatParam 15-1112
15.3.39. ElmoGetAsyncIntArray 15-1113
15.3.40. ElmoGetAsyncFloatArray 15-1114
15.3.41. ElmoSetAsyncArray 15-1115
15.3.42. ElmoGetSyncParam 15-1116
15.3.43. ElmoGetSyncArray 15-1117
15.3.44. ElmoCallAsync 15-1118
15.3.45. ElmoExecute 15-1119
15.3.46. ElmoIsReplyAwaiting 15-1119
15.3.47. ElmoGetReply 15-1120
15.3.48. ConfigVirtualEncoder 15-1121
15.3.49. CancelVirtualEncoder 15-1122
15.3.50. SetPosition 15-1122
15.3.51. SetParameter 15-1123
15.3.52. SetBoolParameter 15-1124
15.3.53. AxisLink 15-1125
15.3.54. AxisUnLink 15-1125
15.3.55. GetBoolParameter 15-1126
15.3.56. GetParameter 15-1126
15.4. The CMMCGroupAxis class 15-1128
15.4.1. SetDefaultParams 15-1130
15.4.2. SetKinTransform 15-1131
15.4.3. RemoveAxisFromGroup 15-1132
15.4.4. MoveCircularAbsolute 15-1133
15.4.5. MoveCircularAbsoluteCenter 15-1135
15.4.6. MoveCircularAbsoluteBorder 15-1136
15.4.7. MoveCircularAbsoluteRadius 15-1137
15.4.8. MoveCircularAbsoluteAngle 15-1138
15.4.9. MoveLinearAbsolute 15-1139
15.4.10. MoveLinearRelative 15-1140
15.4.11. GroupSetOverride 15-1141
15.4.12. GroupSetPosition 15-1142
15.4.13. GroupReadStatus 15-1143
15.4.14. GroupEnable 15-1143
15.4.15. GroupDisable 15-1144
15.4.16. GroupReset 15-1144

15.4.17. GroupReadActualVelocity 15-1145
15.4.18. GroupReadError 15-1146
15.4.19. AddAxisToGroup 15-1147
15.4.20. GroupReadActualPosition 15-1148
15.4.21. GroupStop 15-1149
15.4.22. GroupHalt 15-1150
15.4.23. MoveLinearAbsoluteRepetitive 15-1151
15.4.24. MoveLinearRelativeRepetitive 15-1152
15.4.25. MoveLinearAdditive 15-1153
15.4.26. MovePath 15-1154
15.4.27. PathDeselect 15-1155
15.4.28. PathSelect 15-1155
15.5. The CMMCPPGlobal class 15-1157
15.5.1. RegisterRTE 15-1159
15.5.2. RegisterWarningClbk 15-1159
15.5.3. SetThrowFlag 15-1160
15.5.4. SetThrowWarningFlag 15-1160
15.5.5. SetPrintErrorFlag 15-1161
15.5.6. SetPrintWarningFlag 15-1161
15.5.7. ThrowMessage 15-1162
15.5.8. SetConnectionType 15-1162
15.5.9. SetMessageFileName 15-1163
15.5.10. GetSyncTime 15-1164
15.5.11. SetSyncTime 15-1165
15.5.12. CreateSYNCTimer 15-1166
15.5.13. DestroySYNCTimer 15-1167
15.5.14. GetConnectionReg 15-1167
15.5.15. ConfigBulkRead 15-1168
15.5.16. PerformBulkRead 15-1170
15.5.17. RegisterConnection 15-1171
15.5.18. GetConnectionReg ClearConnectionReg 15-1171
15.6. The CMMCConnection class 15-1172
15.6.1. Event Type Definitions 15-1174
15.6.2. ConnectIPC 15-1176
15.6.3. ConnectIPCEx 15-1176
15.6.4. ConnectRPC 15-1177
15.6.5. ConnectRPCEx 15-1178
15.6.6. SetGlobalBoolParameter 15-1179
15.6.7. GetGlobalBoolParameter 15-1180
15.6.8. GetGlobalParameter 15-1181
15.6.9. SetIsToLoadGlobalParams 15-1182
15.6.10. SetHeartBeatConsumer 15-1183
15.6.11. CallbackFunc 15-1183
15.6.12. RegisterEventCallback 15-1184

15.6.13. RegisterSyncTimerFunction 15-1185
15.7. The CMMCNetwork class 15-1186
15.7.1. GetCommDiagnostics ResetCommDiagnostics 15-1187
15.7.2. ResetCommStatistics 15-1187
15.7.3. GetNetworkInfo 15-1188
15.8. The CMMCHostComm class 15-1189
15.8.1. MbusStartServer 15-1190
15.8.2. MbusStopServer 15-1191
15.8.3. MbusReadHoldingRegisterTable 15-1192
15.8.4. MbusWriteHoldingRegisterTable 15-1193
15.8.5. MbusIsRunning 15-1193
15.8.6. MbusReadCoilsTable 15-1194
15.8.7. MbusWriteCoilsTable 15-1195
15.8.8. MbusReadInputsTable 15-1196
15.8.9. SetModbus[LongSwapped][Short] 15-1197
15.9. The CMMCModbusBuffer class 15-1198
15.10. The CMMCModbusSwapBuffer class 15-1199
15.11. The CMMCNode class 15-1200
15.11.1. Reset 15-1200
15.11.2. ReadStatus 15-1201
15.11.3. SendSDO 15-1202
15.11.4. SendSDODownload 15-1203
15.11.5. SendSDOUpload 15-1204
15.11.6. SendSDOUploadAsync 15-1205
15.11.7. RetreiveSDOUploadAsync 15-1206
15.11.8. PDOGeneralRead 15-1207
15.11.9. PDOGeneralWrite 15-1208
15.11.10. GetPDOInfo 15-1209
15.12. The CMMCBulkRead class 15-1210
15.12.1. CMMCBulkRead 15-1211
15.12.2. Config 15-1212
15.12.3. BulkRead 15-1212
15.12.4. CMMCBulkRead Source Code Examples 15-1213
15.13. The CMMCMotionAxis class 15-1214
15.14. The CMMCPVT class 15-1215
15.14.1. InitPVTTable 15-1216
15.14.2. LoadPVTTable 15-1217
15.14.3. AppendPointsToPVTTable 15-1218
15.14.4. MovePVT 15-1219
15.14.5. UnloadPVTTable 15-1219
15.14.6. CMMCPVT Source Code Examples 15-1220
15.15. The MMCEIPSession class 15-1221
15.15.1. EIPCloseSession 15-1221
15.15.2. EipCreate 15-1222

15.15.3. EipDestroy 15-1222
15.15.4. EipOpenSession 15-1223
15.16. The CMMCEIPDataType class 15-1224
15.16.1. EipTagInit 15-1225
15.16.2. EipSetTag 15-1225
15.16.3. EipGetTag 15-1226
15.16.4. EipCheckReply 15-1227
15.16.5. EipGetData 15-1228
15.16.6. EthernetIP Source Code Examples 15-1229
15.17. TCP/IP and UDP C++ User Libraries 15-1235
15.18. The CMMCUDP class 15-1236
15.18.1. Create 15-1237
15.18.2. SendTo 15-1238
15.18.3. ReceiveFrom 15-1239
15.18.4. SetTimeout 15-1239
15.18.5. ElmoSetAsyncArray 15-1240
15.18.6. ElmoSetAsyncParameter 15-1241
15.18.7. ElmoSetArray 15-1242
15.18.8. ElmoSetParameter 15-1243
15.18.9. ElmoGetAsyncArray 15-1244
15.18.10. ElmoGetAsyncParameter 15-1244
15.18.11. ElmoGetArray 15-1245
15.18.12. ElmoGetParameter 15-1246
15.18.13. ElmoReadData 15-1247

Chapter 16: Appendix 16-1248

16.1. Axis Parameters (Explanations) 16-1248
16.2. Future Supported function Blocks 16-1253
16.2.1. MMC_MoveContinuous 16-1253

Chapter 17: Index 17-1259

Chapter 1: Introduction

This document describes the administrative and motion API of the Elmo Gold Maestro controller. The Gold Maestro – G-MAS – is Elmo's Network Motion Controller. It is a network-based system, which operates in conjunction with Elmo's intelligent servo drive family, to provide a total network motion controller solution.

The G-MAS is designed to support both the existing SimplIQ servo drives, based on standard CAN Open network architecture, as well as the new Gold family, with EtherCAT networking.

As true network controllers Elmo's G-MAS, SimplIQ, and servo drives, share the motion processing workload in distributed motion control architecture. The best servo performances are achieved by combining Elmo's servo drives, and the new real-time motion control capabilities of the G-MAS main controller.

The G-MAS operates as a Network Motion Controller to support:

- Full, Real-Time, Multi-Axis motion synchronization
- Advanced user programming capabilities based on well known standards
- Deterministic control over Motions, IO's, and processes in the system

1.1. G-MAS over (Go) Standard

The G-MAS offers real-time motion control support, for full multi-axis system synchronization, using the industry interface PLCopen for Motion Control standard. This is the G-MAS over standard, applicable to the Gold Line (Gold Maestro), and SimplIQ Line of motion controllers. The use of native C and C++ programming support (run on the G-MAS target) dramatically accelerates execution of user level programs, while maintaining the same PLCopen Motion API definitions as a standard software API.

The operation of C and C++ based programs is optimal, and will result in the best overall system performance, since they generate machine code that runs directly on the target G-MAS hardware processor.

1.2. What the API Does

The purpose of the PLCopen Motion API is to produce a standardized motion control API solution without compromising on system performances, based on the PLCopen Standard. This is achieved by the G-MAS software architecture, since the low-level controller real-time motion engine directly implements the PLCopen motion API, therefore no intermediate layers are required, and performances are optimal.

Software interfaces to the G-MAS Motion API are implemented to the users' convenience, via a dedicated API library. This library supports the following:

- Interfacing the G-MAS PLCopen Motion API from a host computer via Ethernet TCP/IP
- Interfacing the G-MAS PLCopen Motion API from user programs, running on the G-MAS product

An identical API library is used at the Development / Host PC with the API library accessing the same API server to perform the desired user operations.

This same G-MAS API operates with EtherCAT communication and for the SimplIQ family of network controllers.

The user has the ability to store such programs on the G-MAS FLASH, and to run them at power-up. This naturally results in a faster and more optimized method of operating the G-MAS motion API.

1.3. Terminology

The terminology used in this document covers language used throughout the servo drive, controller, and communication industry and are not necessarily specific to Elmo Motion Control Ltd.

Term	Explanation
ACS	Axes Coordinate System: The system of coordinates related to the physical motors.
Axis	Axis is the most basic motion object and is used to control the motion of a single motor/axis.
Blending	A method for consecutive function blocks to cooperate in the transition from the first to the next.
CAN	Controller Area Network. Data link layer protocol for serial communication as specified in ISO 11898-1 (1999).
CiA	CAN in Automation international users and manufacturers group e.V. It is a non-profit association promoting Controller Area Network (CAN).
COB	Communication Object, consisting of one or more CAN frames. Any information transmitted via CANopen has to be mapped into COBs.
COB-ID	COB-Identifier. Identifies a COB uniquely in a CAN network. The identifier also determines the priority of that COB in the data link layer.
CoE	CANopen over EtherCAT. Defines a standard way to access the CANopen protocol and includes an object dictionary, SDO, PDO, and emergency messages.
Contour curve	An inserted curve that modifies the original path. It is the resulting curve after blending.
Coordinate system	The reference system in which a coordinate or path is described.
Corner deviation	The shortest distance between the programmed corner point and the contour curve.
Corner distance	Distance of the start point of the contour curve to the programmed target point.
Direction	The orientation components of a vector in space. Note: This is different from the MC_Direction input.
Drive	A unit controlling a motor via the current and timing in its coils.
EoE	Ethernet over EtherCAT. Fully Ethernet compatible and defines a standard way to exchange or tunnel standard Ethernet frames. Used to create G-MAS Master and Drives as Slaves in EAS and other applications for both diagnostics and download of files.
FB	Function Block
FIFO	First In, First Out. An abstraction in ways of organizing and manipulation of data relative to time and prioritization. This expression describes the principle of a queue processing technique or servicing conflicting demands by ordering process by first-come, first-served (FCFS) behavior: what comes in first is handled first, what comes in next waits until the first is finished, etc.

Term	Explanation
FoE	File over EtherCAT. Similar to TFTP, enables access to any data structure in the device, and defines a standard way to download and upload firmware and other files over the EtherCAT network. Used as a download of hardware configuration files and updates. Refer to the functions 6.3.10 MMC_DownloadFoE, and 6.3.11 MMC_GetFoEStatus
G-MAS	Gold Maestro Application Software also known as the Gold Maestro Network Motion Controller performs synchronized multi axis motions in the system (such as circle, line etc.), using a real time communication protocol so that all drives are synchronized to a specific SYNC signal in the system It operates as a master, independent of any host system. In operational mode, it periodically sends data to the slaves that may override the data that a user sends from a host system.
GoS	G-MAS over SimplIQ
Group	Group of axes
Group-FB	The set of function blocks that can operate on a group of axes.
HPT	High Priority Task as against LPT (Low Priority Task), and MPT (Medium Priority Task), used in Embedded Linux, RTOS, and Parallel Programing for programming in robotics. This refers to the task priority in running threads, which may or may not lock resources.
IO	Input and output
IPC	Inter-process communication (IPC) is a set of techniques to exchange data among multiple threads in one or more processes. Processes may be running on one or more network-linked systems. IPC techniques are divided into methods for passing messages, synchronization, shared memory, and remote procedure calls (RPC). IPC may vary, depending on the bandwidth and communication latency between the threads, and the type of data being communicated. C Programs located on the G-MAS use the IPC method.
Masking	A form of cloaking of communication addresses. The netmask is a bitmask used to separate the bits of the network identifier from the bits of the host identifier. It is written in the same notation used to denote IP addresses.
MCS	Machine Coordinate System: The system of coordinates that is related to the machine. Sometimes called World Coordinate System or Base Coordinate System. With Cartesian built machines, MCS is a Cartesian Coordinate system The coordinate system from the physical multiple axes ACS is linked to the MCS via a kinematic transformation (forward and backward conversion).
Motor	An actuator focused to a movement, converting electrical energy into a force or torque.
Mutex	Mutual exclusion. Mutual exclusion algorithms are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections. A critical section is a piece of code in which a process or thread accesses a common resource. The critical section by itself is not a mechanism or algorithm for mutual exclusion. A program, process, or thread can have the critical section in it without any mechanism or algorithm, which implements mutual exclusion.

Term	Explanation
Orientation	The rotational components of a vector in space.
Path	Set of continuous positions and orientation information in multi-dimensional space. This may be geometrically described as a space curve that the axes group TCP moves along.
PathData	Description of a path, which can include additional information like velocity and acceleration.
PDO	Process Data Object
PDS	Power Drive System
Position	Position means a point in space that is defined by different coordinates. Depending on the used system and transformation, it can consist of up to six dimensions (coordinates), three Cartesian coordinates in space and three coordinates for the orientation. In ACS, there can be even more than six coordinates. If the same position is defined in different coordinate systems, the values of the coordinates are different.
PVT	Position velocity time interpolation mode
RPC	A remote procedure call (RPC) is an inter-process communication allowing a program to initiate a subroutine or procedure to execute in another address space (the G-MAS server) without the programmer explicitly coding the details for this remote interaction. The programmer essentially writes the same code whether the subroutine is local to the executing program, or remote. For example, the EAS uses RPC to communicate with the GMAS.
RPDO	Receive Process Data Object. Communication object of a device, which contains output data.
Scara	A special kinematic for robot or handling applications.
SDO	Service Data Object. Peer-to-peer communication with access to the Object Dictionary of a CANopen device.
Speed	Speed is the absolute value of the velocity without direction.
Synchronization	Combines an axis or axes group (as slave) with an axis as master in order for the slave to execute its synchronized path with the progress of the master, and therefore linked to a single-dimension source for synchronization.
TCP	Tool Centre point, the point in the machine that is commanded to move, typically to the center or the head of the tool. It can be described in different coordinate systems.
TPDO	Transmit Process Data Object. Communication object of a device, which contains input data.
Tracking	Is characterized by an axis group that tracks with its movement, the movement of another axis group.
Trajectory	Time dependent description of the path the TCP of an axes group moves along. Additionally to the geometrical description of the space curve, time dependent state variables like velocity, acceleration, jerk, forces etc. are also specified.
Velocity	For a group of axes this means:

Term	Explanation
	For ACS, the velocities of the different axes. For MCS and PCS it provides the velocity of the TCP.

1.4. How to Use this Document

This document allows a programmer in C to communicate and operate the Gold Maestro Network Motion Controller. It is designed to aid the programmer in setting specific parameters for the appropriate function blocks used by the customer. This section describes how to use the detailed information within each function block, set, and customize its parameters.

The chapters in this document are divided according to the following sections:

Chapter 1: Introduction

Chapter 2: G-MAS Overview, explains the operation modes of the G-MAS

Chapter 3: Function Blocks, describes and details the various single and multiple axes.

Chapter 4: Error Correction Mechanism, describes the mechanism to correct drive position errors, and the functions which are used to apply the correction.

Chapter 5: Bulk Parameters Reading to perform reading of parameters from multiple drives with their relevant functions.

Chapter 6: API Services and Operations, describes the main general function blocks referring to the following:

- Main configuration variables
- Data Recording. Refer to **Chapter 14: Chapter 14:Data Recording on page 14-1038** for further details.
- Resource file uploading and downloading
- Download of new firmware version

Chapter 7: Connectivity and Configuration contains all the Network, Modbus (Host), CANbus (drive), EtherCAT (drive), Interpreter Command, and EtherNETIP communications to the G-MAS server.

Chapter 8: API Events, including the mechanism to handle events in the G-MAS.

Chapter 9: G-MAS Personality, includes detail parameters of the G-MAS itself and the function to create the personality.

Chapter 10: Data Recording. This allows the user to record internal controller variables, store them in local a temporary array, and upload them to a host computer using either one of the controller's communication channels.

Chapter 11: Using C++ functions based on the C functions. These are wrapper functions using similar parameter details as their similarly named C functions.

Chapter 12: Appendix; includes explanation of the Axis Parameters and future supported fucntions.

Chapter 13: Index

Each chapter describes function blocks and their parameters according to the API source files described in **Chapter 1: G-MAS Overview**. Some chapters have specific parameters that are applicable throughout a section and are therefore explained prior to the function block listings for that section. For example, the

chapter **5.4 Axis Status** contains definitions of the **Special Function**, whose variables are sub-functions of the function blocks listed in that section. Certain parameters only apply within a specific function block and their details are recorded after the definitions of that function block. For example, the explanation of **Homing Functions** is detailed after the function block **5.6.2 MMC_Home**.

1.1.1. MMC_Halt

Call this function to command a controlled motion stop for a specific Axis.

```
int MMC_HaltCmd(
        IN MMC_CONNECT_HNDL hConn,
        IN MMC_AXIS_REF_HNDL hAxisRef,
        IN MMC_HALT_IN* pInParam,
        OUT MMC_HALT_OUT* pOutParam
);
```

Motion Mode NC - Supported Distributed - Supported

Source GMAS\includes\MMC_PLCopen_single_API.h

Function Parameters

hConn

Connection handle returned by Init Connection command

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the *MMC_HALT_IN* input structure that receives the HALT Information

pOutParam

Points to the *MMC_HALT_OUT* output structure receiving information as a result of calling the HALT function.

Remarks

The axis is moved to the state Discrete Motion, until the velocity is zero. With the output set to Done, the state is transferred to StandStill.

MMC_Halt is used to stop the axis under normal operation conditions. In non-buffered mode, it is possible to set another motion command during deceleration of the axis, which aborts the MMC_Halt and executes immediately.

If this command is active, the next command can be issued, e.g. a driverless vehicle detects an obstacle and needs to stop. When MMC_Halt is issued, then, before the standstill is reached the obstacle is removed and the motion can be continued by setting another motion command, so that the vehicle continues its motion and does not stop.

Scope

MMC_Halt will only operate from Standstill or Continuous Motion. It will not operate from Stopping, Homing, ErrorStop, or Disabled. Refer to the State diagram in *Figure 6*.

Figure 1-1: Function block layout example

Each function block (**Figure 1-1)** begins a section with its title and a short explanation of the function. The **Description** explains the usage of the function block with the **Scope** describing the conditions for its usage. **Motion Mode** defines whether the function block is supported in NC or Distributed (Non-NC) mode.

NOTE: **Links are highlighted in blue bold.**

Source defines the file source of the function block, with the **Function Parameters** describing the main parameters of the function itself. The logical definition must be retained in order for C to read the parameters of the function block correctly.

Remarks describe the function and its usage in detail, with their respective.

MMC_HALT_IN Structure

```
typedef struct{
        float fDeceleration;
        float fJerk;
        MC_BUFFERED_MODE_ENUM eBufferMode;
        unsigned char ucExecute;
}MMC_HALT_IN;
```

Parameters

fDeceleration

Float value of the deceleration (decreasing energy of the motor). Any positive value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

MC_BufferMode defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is suported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

Buffered mode only applies to Distributed and not NC axis. To use buffered for NC, requires user interface programming.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_HALT_OUT Structure

```
typedef struct{
        unsigned int uiHndl;
        unsigned short usStatus;
        short usErrorID;
}MMC_HALT_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections ***3.3.2 G-MAS Error IDs***, and ***3.3.3 NC Profiler Error IDs on pages 3-35 – 3-39***. Displays an error code as -ve or +ve integers.

Figure 1-2: Function block layout example (cont)

The input and output structures for each function(**Figure 1-2**) display the accepted structure for the input and output with their respective parameters. The parameters and sub-parameters and listed on the left side with their various descriptions opposite, and any references to other sections. The description describes the usage of each parameter and sub-parameter. With sub-parameters, the description defines specific values or enumerator values for the parameters with their explanation and/or a reference to such.

Figure 1 describes the function block for MMC_Halt

Figure 1: MMC_Halt function block

Function Block Code Example

```
int rc;
MMC_HALT_IN     stHalt_in;
MMC_HALT_OUT    stHalt_out;
//
// Inserting the structure parameters:
stHalt_in.fDeceleration       = 100000.0;          // Value of the acceleration
stHalt_in.fJerk                     = 20000.0;            // Value of the Jerk
stHalt_in.eBufferMode         = MC_ABORTING_MODE;  // MC_BufferMode Defines the behavior of the
axis
stHalt_in.ucExecute           = 1;
//
rc = MMC_HaltCmd (hConn, iAxisRef, &stHalt_in, &stHalt_out);
if (rc != 0)
{
    HandleError();
}
```

Implementation Example

The example below shows the behavior of MMC_Halt in combination with MMC_MoveVelocity.

1. A rotating axis is ramped down with MMC_Halt.
2. Another motion command overrides the MMC_Halt command. MMC_Halt allows this, in contrast to MMC_Stop. The axis can accelerate again without reaching standstill.

Figure 2: Combination of two blocks for MMC_Halt – Example

Figure 1-3 diagrammatic explanation and example of a function block

At the end of each function block detailed description (**Figure 1-3)**, is a diagram showing its inputs/outputs. Further information is provided with a real program C code, and implementation, examples of its usage.

Chapter 2: G-MAS Overview

The G-MAS general communication and API architecture is described below in **Figure 2-1**.

Figure 2-1: GoS System Software Structure – Development and Application

The G-MAS supports the following programing interfaces:

Operation Systen	Library
Hosts (TCP/IP)	.NET Library Win32 Library – C and C++ Libraries
Internally (IPC)	C and C++ Libraries IEC 61131-3
VxWorks Host	.NET Library Win32 Library – C and C++ Libraries

The G-MAS API implements direct binary communication interface using the TCP/IP and internal connection shown in **Figure 2-1**. Using the direct binary API is a faster and more efficient connection method to produce best system performances. The underlying Motion API is the PLCopen (the API server), and Elmo's Software API's above it, which export the same functions in both cases. An identical API library is used. Elmo provides the dll/lib for Win32 based environments and a library for interfacing the G-MAS from within the G-MAS, when writing user programs. The only difference is the initial "Initialization" method. The API library accesses the same API server to perform the desired user operations.

The API defines a set of function blocks, with the following attributes:

- Simplicity - ease of use, towards the application program builder and installation & maintenance
- Efficiency - in the number of function blocks, directed to efficiency in design and understanding
- Consistency - conforming to IEC 61131-3 standard
- Universality - hardware independent
- Flexibility - future extensions / range of application

These function blocks are sectioned according to their motion axes, and communication protocols. The API therefore consists of a series of grouped source files divided by the following subjects:

General
Source: GMAS\includes\MMC_general_API.h
Includes all main configuration and firmware download function blocks.

Main Definitions
Source: GMAS\includes\MMC_definitions.h
Includes all main and basic definitions for the function blocks.

Single Axis Motion
Source: GMAS\includes\MMC_PLCopen_single_API.h
Includes administrative and motion function blocks involved in the single axis motion.

Group Axes Motion
Source: GMAS\includes\MMC_PLCopen_group_API.h
Includes administrative and motion function blocks involved in multi-axes motion.

Network Communications
Source: GMAS\includes\MMC_network_API.h
Includes all basic network functions blocks necessary to communicate with the Gold Maestro Network Motion Controller.

Modbus Communications

Source: GMAS\includes\MMC_host_comm_API.h

Includes all Modbus function blocks necessary to communicate with the Gold Maestro Network Motion Controller.

Drive Communications

Source: GMAS\includes\MMC_drive_comm_API.h

Includes all CANbus, EtherCAT function blocks, and Interpreter Command functions necessary to communicate with the Gold Maestro Network Motion Controller.

EtherCAT Communications

Source: GMAS\includes\MMC_ECATIO_API.h

Includes major EtherCAT communication functions for analog and digital I/Os.

DS-401 Communications

Source: GMAS\includes\MMC_DS401_API.h

Includes major DS-401 communication functions for DI and DO intended for I/O modules.

Error Correction

Source: GMAS\includes\MMC_ErrorCorr_API.h

Includes error correction functions for 1D, 2D and 3D modes.

API Events

Source: GMAS\includes\MMC_events_API.h

Includes function blocks that read and write events to and from the G-MAS.

C++ Functions

Source: GMAS\includes\CPP\MMCXXXXXXX.h

Includes all C++ class function mirroring functions and function blocks described in detail for C programming

Figure 2-2: GoS System Software Structure –Host PC Development

In addition, since the G-MAS operates as a master, independant of any host system, in operational mode, it periodically sends data to the slaves, that may override the data that a user sends from a host system. Therefore, for example, the user cannot tune an axis if the axis is in operational mode. To prevent this and allow the Elmo Application Studio (EAS) application (**Figure 2-2**) to operate via the G-MAS CANbus and EtherCAT, specific API functions are called to change the G-MAS operation and allow these applications to function.

The following table lists the function blocks called as part of a special API to allow the EAS application to function and revert the G-MAS to operational mode when their operation is completed:

Application	Function Blocks
EAS	MMC_ChangeToPreOPMode MMC_ChangeToOperationMode GetGMASOperationMode MMC_EnableEthercatConfigMode MMC_DisableEthercatConfigMode MMC_IsEthercatConfigMode

Important: **It should be noted that connecting to the G-MAS is only allowed using one user application at a time. Connecting two applications to the G-MAS in parallel may cause serious problems to the G-MAS library. When performing multiple IPC connections to the G-MAS, the multiple connections must be opened from the same user application.**

2.1. For EAS

For the EAS application to monitor and perform motions, the G-MAS cannot operate in the background. The special API function *MMC_ChangeToPreOPMode* changes the EtherCAT and CANbus communication from the G-MAS to Pre-Operation mode, causing the following:

Communication	Operation
EtherCAT	**No process cycle operates**
CANbus	**No outputs via CAN, and no state machines run**

These API functions change the G-MAS mode so that the G-MAS operation is transparent and no messages transfer between the G-MAS and the drives.

In order to configure the EtherCAT network (EtherCAT Configuration Mode) via the EAS application, the G-MAS must be set to EtherCAT Configuration mode. The user is then able to perform the operations. The API then employs the specific functions to change the G-MAS back to operational mode.

2.2. G-MAS Operation Modes

To optimize the device network usage, the G-MAS supports two modes of operating axes present on the Device Network:

- NC Axes – for Numeric Control Axes
- Distributed – for axes not under strict numeric control

The main difference between these modes is the way the motion profile is calculated, and as a result, the synchronization level achieved.

In general, for axes not requiring low level (network) motion synchronization, the Distributed mode should be used, allowing the servo drives to generate their own motion trajectory, thus reducing network load. In this case, synchronized motions like ECAM, based on an external master encoder can still be executed. For highly synchronized motions, generated by the Master controller (referred to under the PLCopen definitions as group vector motions), the NC mode should be used.

2.2.1. NC Motions

In this mode, the G-MAS controls the motion, handling the axis (and motion) State (as defined by the PLCopen Standard), and calculating the motion profile as part of its real-time loop process (NC Cycle). Servo drives operating with a G-MAS master under this mode will run under the DS-402 motion modes e.g.; Interpolated position, or one of the Cyclic Sync modes (Position/Velocity).

2.2.2. Distributed / Standard DS-402 (stand-alone) Drive

In this mode, the G-MAS uses the servo drives own DS-402 operation modes, where the drive itself controls its own profiling as part of its Real Time process. The G-MAS only synchronizes start/stop and general activation functions, but is not responsible to the low-level real-time profile generation.

The G-MAS can mix NC and Distributed axes in the same network configuration, thus optimizing usage of network and processor resources. The definition of the axis type (NC or Distributed), can be changed during operation using the *ChangeOpMode* (operation mode) command.

2.2.3. G-MAS Axes and Node Definitions

The G-MAS controls the following axis types:

- Single Axis as NC axis
- Single Axis as Distributed axis
- Group of axes, as NC axes (only). A group is a collection of axes, which can execute spatial vector motions

In the G-MAS architecture, all axes names have to be defined in advance, in the system resource file, a dedicated (XML format) file, which defines the following:

- Number of active axes in the system
- For each axis, whether it is operating in NC or Distributed mode.
- Groups of axis must be predefined by the user. However, the link of actual axes to groups can be performed in run-time (using specific API functions, e.g. AddAxisToGroup() and RemoveAxisFromGroup().

- Basic G-MAS network cycle (used for NC as well as Distributed), to access the axis position, commands etc.

For each of the above, the G-MAS hold an internal software object Node. Currently, two types of nodes are defined:

- Single Axis Nodes: NC or Distributed.
- Group (Vector) Nodes: NC only, and can be Group Single axis (NC) only.

Additional Nodes types are supported by the G-MAS, such as DS-401 IO, DS-301, and DS-406 modules.

The Max Axes and Node numbers and their combinations are defined as follows:

Tc The Minimal Time of the Master NC Cycle.

In EtherCAT communication, Tc defines the Minimal Distributed Clock Cycle Time of the system. Its value is currently defined as 1 msec (1000 µs), and the function is defined as:
Tc=(1+N)x250 µsec (where : 1 <= N <= 39).

In CAN Bus communication, Tc defines the Sync Time of the CAN network. Its value is currently defined as 3 msec and the function is defined as:
Tc=Nx1 msec (where 1 <= N <= 100).

N The max number of Single Axis Nodes in the system. Currently limited to 64 axes (NC and Distributed, altogether).

V The max number of Group/Vector nodes that can be simultaneously defined in the system. Currently limited to 16 axes (this is in addition to the 64 single axis nodes).

Va The max number of Group/Vector nodes that can be simultaneously running in the system. Currently limited to 6 Group/Vectors.

Vn The max number of physical axes that can be simultaneously linked to a specific Group/Vector node. Currently limited to 16 physical axes.

Mc The max number of devices that can be accessed in a single Master Cycle, via the Communication link. This number depends if CAN or EtherCAT communications are used. This number also depends on the current selected Tc.

Mp The max number of NC axes that can be handled in a single Master Cycle. This includes both Single axis, as well Vector/Group nodes. Generally speaking, Mc and Mp can be different numbers. Currently, they are equal and limited to 20.

In order to reduce the G-MAS cycle computations, each axis can define an axis period and an axis offset time that is related of course to the Tc base time.

For example: A typical NC system nodes/axes distribution may have:

- 8 Physical axes (nodes: a1 ÷ a8) – all NC.
- 1 group (node v1, linked to physical axes a6, a7, a8).
- Group v1 is running on each master cycle (Tc), calculating a spatial (vector) profiled motion (in 3D space), and the actual projections to the physical axes linked to it are a6, a7, a8.
- The physical axes nodes are running together as follows:
  - a1, a2 are running together, each 2×Tc cycles.
  - a3, a4 are running together, each 2×Tc cycles.
  - a5 is running each 4×Tc cycles.

Figure 2-3: Typical NC Configuration

In this example, the Max Mp is 6 (although we have 8 axes). The cycle is repeating itself once every 4xTc cycles.

2.2.4. G-MAS to Servo Drive Interfaces

The G-MAS manages all motion commands sent to the servo drives, via the CANopen DS-402 standard. This is relevant to the G-MAS CAN hardware interface, and to the EtherCAT protocol implementing CoE (CAN Over EtherCAT).

For axes (Nodes) that operate in NC mode, G-MAS uses the DS-402 motion modes: Interpolated position, or one of the Cyclic Sync modes (Position/Vel).

For axes (Nodes) operating in Distributed mode, where the servo drive manages its own profiler and real-time motion execution, it is assumed that the servo drive supports the relevant requested motion modes.

Motion Modes that are part of the PLCMotion API definition, but are NOT supported by the DS-402 interface, will not be available in standard DS-402 servo drives when working in Distributed mode (unless specific Vendor Types objects are defined, e.g. ECAM in drive level, etc. as implemented for example in Elmo servo drives).

Note: **Although the above describes and relates to Elmo DS-402 compatible servo drives, the G-MAS design does not limit the operation of any DS-402 compatible servo drives as well.**

Chapter 3: G-MAS Hardware and Software Limits Handling

3.1. Introduction

The G-MAS behaves as a Multi Axis Motion controller and regularly updates the system statuses of each drive. However, the user may not wish to see and manage individual drives, only a single interface of the complete system. The disadvantage of this model single interface management, occurs when the behavior of the actual motion of a drive is unexpected, due to the limitations of the drive itself. In this situation the user has no knowledge when an axis or group limit is reached. The profiler continues to maintain its status and handle the axis motion as if no limit exists.

This chapter therefore describes the procedure to manage hardware and software limits to achieve the following:

- Protect the hardware system application from damage or otherwise due to the axis, or group overshooting
- Control the axis position according to the axis /group limits from the G-MAS
- Obtain each axis limit status, either when operating as single axes or group axes

3.2. Interfaces

The limit management mechanism involves interfacing with the following:

- Software axis/group position limits
- Stop parameters
- Status Register
- MCS Limit Register
- Stop on Limit event

3.2.1. Software Position Limits

This uses the presently defined parameters mechanism for single axis/group to determine the limit:

Axis Definition	Limits	
ACS – Axis Level	High Limit MMC_SW_LIMIT_HIGH_POS_PARAM Low Limit MMC_SW_LIMIT_LOW_POS_PARAM	
MCS – Group Level	High Limit array MMC_MCS_SW_LIMIT_HIGH_POS_ARRAY Low Limit array MMC_MCS_SW_LIMIT_LOW_POS_ARRAY	

To describe the difference between High and Low limits the following diagrams display the differences between motion Outside a High or Low Limit and Inside the limits.

Inside High Limit

Inside Low Limit

Out of Limits

3.2.2. Status Register

This varaible provides information on the special status of an axis. The Status Register represents the status of the axis according to the following definitions:

Bit Value	Represents
Bit 0	Reverse Limit Switch (RLS)
Bit 1	Forward Limit Switch (FLS)
Bit 2	ACS Low Limit
Bit 3	ACS High Limit

Bit Value	Represents
Bit 4	MCS Low limit
Bit 5	MCS High limit
Bit 6\7\8\9	for internal use

To verify the current limit, use one of the following methods:

- Check the function **MMC_GetStatusRegister** (refer to section **10.3.21** fo rthe function details) and look at the uiStatusRegister variable value
- For the Bulk Read mechanism, use presets 4 or 5
- For the Recording mechanism, use the vector NC_STATUS_REGISTER = 66

3.2.3. ACS\SingleAxis

The following bits are relevant for ACS limits:

Register Bit Value
Bit0 – RLS
Bit1 - FLS

Register Bit Value
Bit2 - Low
Bit3 – High

3.2.4. MCS

The following bits are relevant for MCS limits:

Bit Value	Represents
Bit 4	MCS Low limit
Bit 5	MCS High limit

3.2.5. MCS Limit Register (32 bits)

This parameter represents the status of the MCS limits in a specific group. To obtain this parameter, use one of the following methods:

- Check the function **MMC_GetStatusRegister** (refer to section **10.3.21** fo rthe function details) and look at the uiStatusRegister variable value
- For the Bulk Read mechanism (refer to the **Chapter 8: Bulk Parameters Reading**), use the Status Register with presets 4 or 5
- For the Recording mechanism (refer to the **Chapter 14: Data Recording**), use the vector NC_STATUS_REGISTER = 67

Register Bit Value	Register Bit Value	Register Bit Value	Register Bit Value	Register Bit Value
Bit0 – X Low	Bit7 - U High	Bit14 – N2 Low	Bi21 – N5 High	Bit28 – N9 Low
Bit1 - X High	Bit8 – V Low	Bit15 – N2 High	Bit22 – N6 Low	Bit29 – N9 High
Bit2 - Y Low	Bit9 - V High	Bit16 – N3 Low	Bit23 – N6 High	Bit30 - S Low
Bit3 - Y High	Bit10 - W Low	Bit17 – N3 High	Bit24 – N7 Low	Bit31 - S High
Bit4 – Z Low	Bit11 - W High	Bit18 – N4 Low	Bit25 – N7 High	
Bit5 – Z High	Bit12 – N1 Low	Bit19 – N4 High	Bit26 – N8 Low	
Bit6 - U Low	Bit13 – N1 High	Bit20 – N5 Low	Bit27 - N8 High	

3.2.6. Stop Parameters

These parameters used to define the kinematic of the stop procedure when the limit is detected. To obtain and set these parameters, use the G-MAS parameters mechanism and the following specific parameters:

- MMC_LIMIT_STOP_DECELERATION (57)
- MMC_LIMIT_STOP_JERK (58)

These parameters can be applied for both, single axis or group.

3.2.7. Stop-on-Limit Event

The special event Stop-on-Limit is created at application level (when the event is registered) to supply the following data:

- Stop\Group_stop error code
- Axis\Group reference
- Status Register
- MCS Limits Register

As a result, an applicable error code is created when that Stop-on-Limit was unsuccessful. Otherwise the Stop/Group error code should return 0.

3.3. Function Block Pre-Insertion Behavior

The behavior of a function block prior to its insertion in the operational queue varies between single and multiple grouped axes.

3.3.1. Single Axis

For a single axis, the function block will automatically check whether the software limits have been reached, and if reached, will then check the allowed direction for the function block operation on the axis.

NOTE: **For optimal performance in Distributed mode (CAN), configure the PDO with the actual position.**

When the hardware limit is reached in Distributed mode, the state of the axis changes to StandStill.

The following table describes the alternative behavious for any single axis function block type described in this document.

Inserted FB type for Single Axis			
Current state	**Admin FB**	**NC - Motion FB**	**NonNC - Motion FB**
Out of limit	Always - √	If the FB is one of this: MoveAbs. MoveAbsRep. MoveRel. MoveRelRep. MoveAdd. MoveAddRep. MoveVel. Target position outside limit - √ Target position inside limit - X For all other FB's – always √	If the FB is one of this: MoveAbs. MoveAbsRep. MoveRel. MoveRelRep. Target position outside limit - √ Target position inside limit – X For all other FB's – always √
Inside limit	Always - √	If the FB is one of this: MoveAbs. MoveRel. MoveAdd. MoveVel. **HW limit** If RLS – positive direction - √ If RLS – negative direction - X If FLS – positive direction - X If FLS – negative direction - √ **SW limit** If Low limit – positive direction - √ If Low limit – negative direction - X If High limit – positive direction - X If High limit – negative direction - √	If the FB is one of this: MoveAbs. MoveRel. **HW limit** If RLS – positive direction - √ If RLS – negative direction - X If FLS – positive direction - X If FLS – negative direction - √ **SW limit** If Low limit – positive direction - √ If Low limit – negative direction - X If High limit – positive direction - X If High limit – negative direction - √ For all other FB's – always X

Inserted FB type for Single Axis			
		For all other FB's – always X	

3.3.2. Multi Axes Group

For a multi-axes group, the function block will automatically check whether the software limits have been reached for each axis in the group. If reached, will then check the allowed direction for the function block operation on each problematic axis in the group.

If the function block is inserted in ACS mode, the actual position is checked against the ACS software limits in the axis level (for further details on the definitions of kinematic transformations, refer to the section **5.8.1 Coordinate System and kinematic transformation**).

If in MCS mode, the desired position of each kinematic direction is checked against the MCS software limits in the group level.

The following table describes the alternative behavious for any multiple-axes group axis function block type described in this document.

Inserted FB type for Multi Axes		
Current state	Admin FB	Motion FB
All axes Out of limit	Always - √	If the FB is one of this: MoveLinearAbsolute. MoveLinearAbsoluteRepetitive. MoveLinearRelative. MoveLinearRelativeRepetitive. MoveLinearAdditive. MoveCirculatAbsolute Target position outside limit - √ Target position inside limit - X For all other FB's – always √
One or more axes Inside limit	Always - √	If the FB is one of this: MoveLinearAbsolute. MoveLinearRelative. MoveLinearAdditive. **HW limit (per groups member)** If RLS – positive direction - √ If RLS – negative direction - X If FLS – positive direction - X If FLS – negative direction - √ **SW limit (per groups member)** If Low limit – positive direction - √ If Low limit – negative direction - X

Inserted FB type for Multi Axes		
		If High limit – positive direction - X If High limit – negative direction - √ For the following FB's – always X MoveLinearAbsoluteRepetitive. MoveLinearRelativeRepetitive. MoveCirculatAbsolute For all other FB's – always √

3.4. Real Time Behavior

How regularly is the hardware and software limit 32bits Status Register updated? The answer is generally every cycle. For single axis or ACS group motion, the hardware and software limits (ACS bits, 0-3 bits) are checked every cycle. However, for the MCS group, the MCS software limits (4-5 bits) are only tested when the group is in motion. The following table summarise this behaviour:

MCS Software Limits	When checked?
Single Axis	never
Multi Axis (ACS)	never
Multi Axis (MCS)	when in motion

The limit bits in the Group Status Register are only updated when the group is not in Disabled state.

When the limit is detected, and the axis or group is in motion, the motion will be stopped immediately. Then a Stop event is sent to the application level provided that the Stop event is registered. After the Stop is completed (if in motion), the position of all relevant axes are synchronized.

However, for Distributed (Non NC) axes, when the limit is detected, the motion is not stopped from the G-MAS, but reverts to the servo drive, which will manage the stopping of the axis. Then a Stop event is sent to the application level provided that the Stop event is registered.

3.5. System Constraints And Limitations

The hardware limit recognition only operates under the two following conditions:

- RLS and FLS data representatives are mapped on the drive level
- Digital Input object is mapped to the G-MAS

If both these conditions are not accomplished, the G-MAS cannot manage the hardware limits. However, if the conditions are fulfilled, but the user wishes to disable the servo drive management of the limits, then set the command **XA[4]** on the Gold servo drive (only Gold drives support this command).

This command disables the following:

Bit	Disabled
Bit 0	bypass position software limit
Bit 1	bypass acceleration limit (Stop Deceleration(SD))
Bit 2	bypass hardware limit

The profiler is normally synchronized with the actual position of the drive, when the limit is detected. However, a situation may occur when the limit is detected, the profiler and the drive position are mismatched due to a different stop deceleration on the G-MAS and the drive. This may cause the drive to unexpectedly jump position on the next motion.

This situation is managed, as described above, by synchronizing the position of all relevant axes. Refer to the two graphs below for an example.

SD < GMAS_DC

SD > GMAS_DC

The G-MAS is not responsible for unexpected behavior when the Drive Software limits are lower than Software Limit of the G-MAS.

Drive Limit is wrong

The following should be noted:

- The Software Limits can be changed during the axis or group motion.
- There is no protection against non-motion deadlocks:
  - RLS & HighLimit
  - FLS & LowLimit

When the limit is reached, buffered or blended mode is forbidden. In addition, for Distributed mode, the stop event will be sent when the command XA[4] = 7.

Chapter 4: G-MAS Initial Connection

4.1. G-MAS USB Connection

This section describes how to communicate with the G-MAS when connected via a USB connection from a host system to allow the GMAS ports to be configured and operate via the LAN. This connection imitates a COM port connection at a COM Port. Therefore, if an RS232 Terminal connection is opened at the host system, *udev* execution opens a terminal from which it is possible to perform the following basic operations.

- Change / Read IP Address.
- Change / Read Gateway.
- Change / Read Subnet mask.
- Change / Read Server IP.
- Change / Read download version path.
- Change / Read DHCP

4.1.1. Connection Procedure

The following procedure describes how to connect the host system to the G-MAS via USB connection and configure the communication parameters of the G-MAS.

1. Make sure that the G-MAS is powered on.
2. Connect the USB connection from the host system to the G-MAS. The G-MAS should emit a sound signifying that a connection is made.
3. Open the Device Manager and locate the Ports section in the hierarchal structure. Verify which COM port is defined for the Elmo GMAS.
4. At the host computer, open a communications Terminal to a COM port.
   The prompt should display *GMAS>*.

5. At the prompt, enter any command detailed in sections **4.1.2** - **4.1.5** to perform the appropriate operation at the G-MAS.

   For example; To request the IP address enter *ipaddr*.

   The G-MAS IP address is returned.

Note: **By default, the G-MAS IP address is set to 192.168.1.3. However, the customer may prefer to integrate the G-MAS with his network system and therefore may wish to change the default value. Use this procedure to perform this action.**

4.1.2. dhcp – Static / Dynamic IP

Purpose	1. **Set DHCP mode.** 2. **Request display DHCP mode.**			
Syntax	dhcp			
Parameters	None or integer			
Attributes				
	Parameter, integer	Interpreter	N/A	0 – Static IP 1 – Dynamic IP

Examples

	Input	**Output**
	`dhcp`	`Static IP`
	`dhcp1`	`OK`
	`dhcp`	`Dynamic IP`

4.1.3. ipaddr – GMAS IP Address

Purpose	1. Set a new IP address. 2. Request display of GMAS's IP address.			
Syntax	ipaddr			
Parameters	None or string			
Attributes				
	Parameter, string	Interpreter	N/A	N/A

Examples

	Input	Output
	`ipaddr`	`10.10.10.1`
	`Ipaddr 10.10.20.2`	`OK`

4.1.4. ipmask – GMAS IP Subnet Mask

Purpose	1. Set a new IP subnet mask. 2. Request display of GMAS's IP subnet mask.			
Syntax	ipmask			
Parameters	None or string			
Attributes				
	Parameter, string	Interpreter	N/A	N/A

Examples

	Input	Output
	`ipmask`	`255.255.255.0`
	`ipmask 255.255.255.0`	`OK`

4.1.5. defgateway – GMAS Default Gateway

Purpose	1. Set a new default gateway. 2. Request display of GMAS's default gateway.			
Syntax	defgateway			
Parameters	None or string			
Attributes				

	Parameter, string	Interpreter	N/A	N/A

Examples

	Input	Output
	`defgateway`	`10.10.10.1`
	`Defgateway 10.10.10.2`	`OK`

4.2. G-MAS PC configuration

4.2.1. XP Windows Setup

To set the PC configuration in XP Windows

1. Select **Start**-> **Control Panel**->**Network Connections**->**Local area connection**->**Properties**

2. In the General tab select Internet Protocol(TCP/IP).

3. Perform the USB Connection procedure as described in the section above **4.1.1.**
4. From the terminal window, check the IP Address, Default Gateway, and Subnet Mask, of the G-MAS.

5. Click Properties and select the radio button *Use the following IP Address*.

6. Enter the IP Address, Default Gateway, and Subnet Mask from the terminal window.
7. Click **OK**.
8. Open the EAS application.
9. Click System Configuration

10. In the Connection type select **GMAS TCP/IP**.

11. In **IP Address** select the same addresses entered to the Internet Protocol Properties window above.

12. Click **Apply** to save the drive's properties.

13. Right click the drive's name and click **Connect.** The GMASResource data window opens.

14. In the GMASResource, select *Upload Resource file from GMAS and update Workspace*. The Resource file is uploaded to the host system.

On completion the G-MAS is connected (green marker), and drives connected to the G-MAS appear.

15. Select the first drive and right-click to choose **Connect**. The drive Personality is uploaded, and the drive is connected. Connect the other drives linked to the G-MAS.

4.2.2. Windows 7 Seup

To set the PC configuration in Windows 7

1. Connect the USB connection from the Host system to the G-MAS.
2. Perform the USB Connection procedure as described in the section above **4.1.1.**
3. From the terminal window, check the IP Address, Default Gateway, and Subnet Mask, of the G-MAS.

4. Open the Network Connection window, and locate the Local Area Connection to the G-MAS.

5. Right-click on the Connection, and select **Status**. Make sure that the IPv4 and IPv6 Connectivity show *No network access*.

6. Click **Properties**, and select *Internet Protocol Version 4 (TCP/IPv4)*.

7. Select **Properties** and enter the Default Gateway, and Subnet Mask obtained from the Telnet window.

8. In the Internet Protocol Version 4 (TCP/IPv4 Properties window, Insert an **IP Address** similar to the G-MAS own address but different at the fourth set of digits, as shown e.g. 192.168.1.2, and then click **OK**.
9. In the Local Area Connection Properties window, select *Internet Protocol Version 6(TCP/IPv6)*.
10. In the Internet Protocol Version 6(TCP/IPv6) Properties window, click the radio button next to *Obtain an IPv6 address automatically* as shown below, and click **OK**.

11. Check the connection to the G-MAS by pinging it. Enter the following at the Windows prompt:
Ping –t <G-MAS IP Address> e.g.192.168.1.3

A Command Prompt should open with the reply results demonstrating a connection to the G-MAS.

12. Open the EAS Application at the System Configuration window and right-clcik on the Workspace to setup a new G-MAS.

13. Setup the EtherCAT configuration of the G-MAS with respective TCP/IP addresses setup in the Windows configuration and from the Terminal feedback, and click .

14. Click **Apply** to save the drive's properties.

15. Right click the drive's name and click **Connect.** The GMASResource data window opens.

16. In the GMASResource, select *Upload Resource file from GMAS and update Workspace*. The Resource file is uploaded to the host system.

On completion the G-MAS is connected (green marker), and drives connected to the G-MAS appear.

17. Select the first drive and right-click to choose **Connect**. The drive Personality is uploaded, and the drive is connected. Connect the other drives linked to the G-MAS.

Chapter 5: Motion and Administrative Function Blocks

The G-MAS supports the Single Axis (including synchronized master/slave) and group coordinated motions with their related administration function blocks.

The following subsections describe all the function blocks, their usage with examples of their implementation.

5.1. Compliance and Portability

As part of the PLCopen specification, the function blocks are designated specific abbreviations in the function block descriptions, depending on whether the function is described as:

- Basic input and output variables are marked as B in the function block definitions.
- For higher-level systems and future extensions any subset of the extended input and output variables, are marked as E.
- Vendor specific additions are marked with V.

5.2. Function Block States

5.2.1. Single Axis

Figure 5-1: Single axis function block states

Note the following in **Figure 5-1**:

1. From any state. An error in the axis occurred.

2. From any state. MMC_Power.Enable = FALSE and there is no error in the axis.
3. MMC_Reset AND MMC_Power.Status = FALSE
4. MMC_Reset AND MMC_Power.Status = TRUE AND MMC_Power.Enable = TRUE
5. MMC_Power.Enable = TRUE AND MMC_Power.Status = TRUE
6. MMC_Stop.Done = TRUE AND MMC_Stop.Execute = FALSE

The axis is always in one of the defined states described in **Figure** 5-1. A change of state is reflected immediately when issuing the corresponding motion command. Any motion command that initiates a transition changes the state of the axis and consequently, modifies the way the current motion is computed. The above state diagram is an abstraction layer of the axis's real state, comparable to the image of the I/O points within a cyclic (PLC) program.

Note: **The response time Immediate is system dependent, coupled to the state of the axis, or an abstraction layer in the software.**

Figure 5-1 is focused on a single axis. Arrows within the state diagram show the possible state transitions between the states. State transitions due to an issued command are shown by full arrows. Dashed arrows are used for state transitions that occur when a command of an axis has terminated or a system related transition (like error related). The motion commands, which transit the axis to the corresponding motion state, are listed above the states. These motion commands may also be issued when the axis is already in the according motion state.

In general, for all states, the axis state can be read using the command MMC_ReadStatus with the following applicable definitions:

Done Signals that the function block has perform its operation.

Valid A valid set of outputs is available at the function block.

Disabled The state Disabled describes the initial state of the axis. In this state, the movement of the axis is not influenced by the function blocks. Power is Off and there is no error in the axis. If the MMC_Power function block is called with Enable=TRUE while being in Disabled, the state changes to Standstill. The axis feedback is operational before entering the state StandStill. Calling MMC_Power with Enable=FALSE in any state except ErrorStop transfers the axis to the state Disabled, either directly or via any other state. Any on-going motion commands on the axis are aborted (CommandAborted).

Error Signals that an error has occurred within the function block. This may be a communication or axis (servo drive) error.

ErrorStop ErrorStop is valid as highest priority and applicable in case of an error. The axis can be either power enabled or disabled and can be changed via MMC_Power. However, as long as the error is pending the state remains in ErrorStop. The intention of the ErrorStop state is to bring the axis to stop, if possible. There are no further function block commands accepted until the reset command, MMC_Reset has been performed from the ErrorStop state. The transition to ErrorStop refers to errors from the axis and axis control, and not from the function block instances. These axis errors may also be reflected in the output of the

	function block instances errors.
ErrorID	Error identification
StandStill	Power is on, there is no error in the axis, and there are no motion commands active on the axis. The axis is not in motion.
Busy	Shows that the function block is operating and new output values can be expected.
Stopping	Refer to **Figure** 5-1.
Discrete Motion	Refer to **Figure** 5-1
Continuous Motion	Refer to **Figure** 5-1
Homing	Refer to **Figure** 5-1.

5.2.2. Group of Axes

The state-diagram of the group describes the commanded state of a group of axes, and is supplementary to the single axis state diagram. Therefore, where axes are in a group state, the single axis state diagram is also active per axis and the two state diagrams are interdependent. The basic group states are:

GroupDisabled	The initial state at power up where a group can be created. Issuing *MMC_GroupEnable* cancels this state.
GroupStandby	In this state, the group is enabled and no function block has control on one of the axes in the group. In this state, the group can additionally be altered and homed if needed (State *GroupHoming*).
GroupMoving	If a function block has control on (one of the axis of) the group, the state changes to *GroupMoving*.
GroupStopping	A special state that deals with the *MMC_GroupStop* command, which automatically transfers to the state *GroupStandby* as soon as Done is SET and Execute is FALSE in MMC_GroupStop.
GroupErrorStop	In case an error arises (in one of the axis), the state changes to *GroupErrorStop*, which can only be left via issuing *MMC_ResetGroup*.

The Group motion commands will always lead to a synchronized motion state in the single axis state diagram. In case of a *GroupStandby*, all axes of the group are also in single axis state *StandStill*.

A *GroupErrorStop* will not lead to *ErrorStops* of the grouped axes as the error may only affect the group. In case of a single axis ErrorStop, the Group will also change to *GroupErrorStop* as the single error effects the group.

Figure 5-2: Group states

The following notes apply to **Figure 5-2** above.

1 Applicable for all non-administrative (moving) function blocks.

2 In the states GroupErrorStop or GroupStopping, all Function Blocks can be called, although they will not be executed, except MMC_GroupReset for GroupErrorStop and any occurring Error– they will generate the transition to GroupStandby or GroupErrorStop respectively

3 MMC_GroupStop.DONE AND NOT MMC_GroupStop.EXECUTE

4 Transition is applicable if last axis is removed from the group

5 Transition is applicable while group is not empty.

6 MMC_GroupDisable and MMC_RemoveAxisFromGroup can be issued in all states and will change the state to GroupStandby

5.2.3. Function Block Status Bit Masks

The following table list the function block status bit masks for Single axis motion.

Bitwise value	Parameter	Explanation of parameter
0x0200	MC_FB_CMD_ABORT_BIT_MASK	Function Block Command Abort Bit Mask. Vendor Defined. It indicates that the function block has received an abortive command.

Bitwise value	Parameter	Explanation of parameter
0x0100	MC_FB_BUSY_BIT_MASK	Function Block Busy State Bit Mask.
0x0080	MC_FB_ACTIVE_BIT_MASK	Function Block Active State Bit Mask.
0x0040	MC_FB_DONE_BIT_MASK	Function Block Done State Bit Mask.
0x0020	MC_FB_ABORTED_BIT_MASK	Function Block Aborted State Bit Mask.
0x0010	MC_FB_ERROR_BIT_MASK	Function Block Error State Bit Mask.
0x0008	MC_FB_IN_GEAR_MASK	Function Block In Gear State Bit Mask.
0x0004	MC_FB_IN_SYNC_MASK	Function Block In Sync State Bit Mask.
0x0002	MC_FB_IN_VELOCITY_MASK	Function Block In Velocity State Bit Mask.
0x0001	MC_FB_END_OF_PROFILE_MASK	Function Block End Of Profile State Bit Mask.

5.3. Axis, Group, Global, Parameters

The axis, group, global, parameters define the MMC_PARAMETER_LIST_ENUM eParameterNumber values. The parameters define the values of the Axis, Group, Global Status in the following function blocks:

MMC_ReadBoolParameter	MMC_GlobalWriteBoolParameter
MMC_ReadParameter	MMC_GlobalWriteParameter
MMC_ReadStatus	MMC_GroupReadParameter
MMC_WriteBoolParameter	MMC_GroupReadBoolParameter
MMC_WriteParameter	MMC_GroupWriteBoolParameter
MMC_GlobalReadBoolParameter	MMC_GroupWriteParameter
MMC_GlobalReadParameter	MMC_GroupReadStatus
MMC_WriteGroupOfParameters	MMC_ReadGroupOfParameter

The functions MMC_WriteGroupOfParameters and MMC_ReadGroupOfParameter have the following advantages:

- Option to Read /Write up to five sets of parameters in a single function, instead of using five functions to Read/Write a single parameter.
- The user is not required to be concerned whether the parameter is global, specific, or real. These functions will handle all types.
- The function MMC_WriteGroupOfParameters can be inserted into a function block queue, to be performed at a later stage, rather than immediately.

The parameters are subdivided into four separate tables, which define the Boolean and Real aspects of the Global and Specific Axis/Vector parameters as shown in **Figure 5-3** below. These are listed in **Parameters Tables**

Table 1 - 6. For a full explanation of these parameters, Refer to the section **16.1 Axis Parameters (Explanations) on page 16-1248**. These tables will be updated periodically.

Figure 5-3: Division of the axis parameters

Use the following legend to explain the letter symbols used in the Parameters table heading.

5.3.1. Legend

Symbol	Explanation
A	Parameter return value type, with possible values: BOOL - 1, REAL - 2, Undefined – 0 Boolean is defined as an integer, whereas Real is a floating point number
B	Parameter Type – bitwise with options: 1 Global variable (first bit) 2 Axis related (second bit) 3 Group related (third bit)
C	Parameter default value ($Y \times 10^Z$)
D	Parameter minimum value ($Y \times 10^Z$)
E	Parameter maximum value ($Y \times 10^Z$)
F	Permission – bitwise, with user values 0 - 6, and all other bits for internal uses. Parameters are Read/Write and/or Save to FLASH on G-MAS: 0: Read only (first bit) 1: Read/Write (second bit) 2: Save parameters to file in FLASH (third bit) 3: Internal use 4: Internal use 5: Internal use 6: The parameter is not supported in queued write (WriteGroupOfParameters Function). The Parameters are written and saved to the G-MAS as long as the G MAS is powered On. When switched Off, the Parameters are no longer written or saved.
G	Real type of parameter pointer
H	Array size – if 0, not array. If not zero, size of array.

5.3.2. Parameters Tables

Table 1: Boolean Global Parameters table

Index	Parameter description	A	B	F	D	E	C
6	Not in Use						
8	Not in Use						
9	Not in Use						
29	CYCLE_TIME	0	1	5	0	1E+06	1000
30	RES_ID	0	1	5	0	1E+06	0
49	CONNECTION_TYPE	0	1	5	1	2	2
53	SUPPORT_BLENDED_FB_PENDING	0	1	6	0	1	0

54	FB_CNT_BEFORE_ACTIVE_FB_ON_END_VELOCITIES_RECALCULATION	0	1	6	0	10	0
55	ESTIMATED_TIME_TOBE_ACTIVE_FB_ON_END_VELOCITIES_RECALCULATION	0	1	6	0	10000	60

Table 2: Boolean Axis Parameters table

Index	Parameter description	A	B	F	D	E	C
1	AXIS_MODE	0	2	5	0	1	0
2	OP_MODE	0	2	5	0	1	0
3	AXIS_STATE	0	2	5	0	1E+06	1E+06
4	GROUP_ID	0	2	5	0	128	128
5	DRIVE_ID	0	2	5	0	128	128
14	ACT_VEL	2	2	13	0	1E+06	0
17	IN_CONST_VEL	0	2	13	0	1	0
21	IN_AC	0	2	21	0	1	0
25	IN_DC	0	2	21	0	1	0
33	COMM_I_EV_USR_1	0	2	6	-2E+09	2E+09	0
34	COMM_I_EV_USR_1_AUX	0	2	6	-2E+09	2E+09	0
37	COMM_I_EV_USR_2	0	2	6	-2E+09	2E+09	0
38	COMM_I_EV_USR_2_AUX	0	2	6	-2E+09	2E+09	0
41	COMM_I_EV_USR_3	0	2	6	-2E+09	2E+09	0
42	COMM_I_EV_USR_3_AUX	0	2	6	-2E+09	2E+09	0
45	COMM_I_EV_USR_4	0	2	6	-2E+09	2E+09	0
46	COMM_I_EV_USR_4_AUX	0	2	6	-2E+09	2E+09	0
50	ERROR_CORRECTION_PARAM	0	2	5	0	100	0
52	MOTOR_ON_CMD_MAX_TIMEOUT_MS	0	2	38	100	40000	7000
56	MMC_MAX_CURRENT_PARAM	0	2	6	0	65535	2000

Table 3: Boolean Group Parameters table

Index	Parameter description	A	B	F	D	E	C
4	GROUP_ID	0	3	5	0	128	128
28	SPATIAL_OPTION	0	4	6	0	1	0

Table 4: Real Axis Parameters table

Index	Parameter description	A	B	F	D	E	C
10	SET_POS	2	2	5	0	1E+08	0
11	ACT_POS	2	2	5	0	1E+08	0
12	AIM_POS	2	2	5	0	1E+08	0

Index	Parameter description	A	B	F	D	E	C
13	SET_VEL	2	2	13	0	1E+06	0
35	COMM_F_EV_USR_1	2	2	6	-2E+09	2E+09	0
36	COMM_F_EV_USR_1_AUX	2	2	6	-2E+09	2E+09	0
39	COMM_F_EV_USR_2	2	2	6	-2E+09	2E+09	0
40	COMM_F_EV_USR_2_AUX	2	2	6	-2E+09	2E+09	0
43	COMM_F_EV_USR_3	2	2	6	-2E+09	2E+09	0
44	COMM_F_EV_USR_3_AUX	2	2	6	-2E+09	2E+09	0
47	COMM_F_EV_USR_4	2	2	6	-2E+09	2E+09	0
48	COMM_F_EV_USR_4_AUX	2	2	6	-2E+09	2E+09	0
57	MMC_LIMIT_STOP_DECELERATION	2	2	6	0	1E+12	1E+12
58	MMC_LIMIT_STOP_JERK	2	2	6	0	1E+15	1E+15
63	MMC_TARGET_RADIUS	2	2	6	0	1E+51	10000
64	MMC_TARGET_TIME	2	2	6	0	6000	0
65	MMC_PROFILE_TIME	2	6	5	0	400000	0
66	MMC_OVERALL_MOTION_TIME	2	6	5	0	400000	0
67	MMC_MAX_TRACKING_ERROR_POSITION	2	2	6	0	1E+51	10000
68	MMC_MAX_TRACKING_ERROR_TIME	2	2	6	0	6000	0
69	MMC_DRIVE_TRACKING_ERROR	1	2	5	-2147483648	2147483647	0
70	MMC_GMAS_TRACKING_ERROR	2	2	5	-1E+51	1E+51	0
71	MMC_END_MOTION_REASON	1	6	5	0	256	0
72	MMC_AXIS_ERROR_MASK	1	2	5	0	65535	0
73	MMC_LAST_DRIVE_EMERGENCY	1	2	5	0	65535	0
74	MMC_AXIS_ASYNC_ERROR_CODE	1	2	5	0	65535	0
75	MMC_FB_DEPTH	1	6	5	0	1000	0
76	MMC_ANALOG_INPUT	1	2	5	0	65535	0
77	MMC_EMCY_SET_ERR	1	2	6	0	1	1
78	MMC_ETHERCAT_DRV_OUTPUT	1	2	6	0	4294967295	0

Table 5: Real Global Parameters table

Index	Parameter description	A	B	F	D	E	C
7	Not in Use						
59	MMC_SET_VECTOR_VELOCITY_PARAM	2	4	5	0	1E+12	0
62	MMC_MCS_S_DIRECTION	2	4	6	-1E+51	1E+51	0

Table 6: Real Group Parameters table

Index	Parameter description	A	B	F	D	E	C	H
15	MAX_VEL	2	3	14	-1E+11	1E+11	1E+11	
16	Not in Use							
18	SET_AC	2	3	21	0	4E+08	0	
19	MAX_AC	2	3	22	0	1E+12	1E+12	
20	SW_MAX_AC	2	3	18	0	1E+12	1E+12	
22	SET_DC	2	3	21	0	4E+08	0	
23	MAX_DC	2	3	22	0	1E+12	1E+12	
24	Not in Use							
26	MAX_JERK	2	3	30	0	1E+15	1E+15	
27	Not in Use							
31	MMC_SW_LIMIT_HIGH_POS_PARAM	2	2	6	-1E+51	1E+51	1E+51	
32	MMC_SW_LIMIT_LOW_POS_PARAM	2	2	6	-1E+51	1E+51	-1E+51	
51	S_FACTOR_FOR_POLYNOMIAL_TRANSITION	2	3	6	0	10	1.4	
60	MMC_MCS_SW_LIMIT_LOW_POS_ARRAY	2	3	6	-1E+51	1E+51	-1E+51	16
61	MMC_MCS_SW_LIMIT_HIGH_POS_ARRAY	2	3	6	-1E+51	1E+51	1E+51	16

5.4. Axis Status

For specific function blocks the Axis Status and Bit Mask is defined. The axes status bit mask derives from 4 btye group within the G-MAS allocated to this mask separated into the various Common, Group and Single axes, parameters as shown in **Figure 5-4**.

Figure 5-4: Status Bit Mask

The following table describes the Axis Status Bit Masks and their values. These are the outputs of the variable **ulState** related to the axis state (returned in the MMC_ReadStatus function block):

Bitwise value	Parameter	Explanation of parameter
0x80000000	NC_AXIS_DONE_MASK	Axis/Group Motion Done Bit Mask
0x40000000	NC_AXIS_VALID_MASK	Axis/Group Data Valid Bit Mask
0x20000000	NC_AXIS_ERROR_MASK	Axis/Group Error Bit Mask
0x10000000	NC_AXIS_PENDING_MASK	Axis/Group Pending Bit Mask
0x00000800	NC_AXIS_BUSY_MASK	Axis Busy State Bit Mask
0x00000400	NC_AXIS_ERROR_STOP_MASK	Axis Error Stop State Bit Mask
0x00000200	NC_AXIS_DISABLED_MASK	Axis Disabled State Bit Mask
0x00000100	NC_AXIS_STOPPING_MASK	Axis Stopping State Bit Mask
0x00000080	NC_AXIS_STAND_STILL_MASK	Axis Stand Still State Bit Mask
0x00000040	NC_AXIS_DISCRETE_MOTION_MASK	Axis Discrete Motion State Bit Mask
0x00000020	NC_AXIS_CONTINUOUS_MOTION_MASK	Axis Continuous Motion State Bit Mask
0x00000010	NC_AXIS_SYNCHRONIZED_MOTION_MASK	Axis Synchronized Motion State Bit Mask

0x00000008	NC_AXIS_HOMING_MASK	Axis Homing State Bit Mask
0x00000004	NC_AXIS_CONSTANT_VELOCITY_MASK	Axis Constant Velocity State Bit Mask
0x00000002	NC_AXIS_ACCELERATING_MASK	Axis Accelerating State Bit Mask
0x00000001	NC_AXIS_DECELERATING_MASK	Axis Decelerating State Bit Mask

The following table is the of status bit masks for multiple axes.

Bitwise value	Parameter	Explanation of parameter
0x00020000	NC_GROUP_STANDBY_MASK	Group Standby State Bit Mask
0x00010000	NC_GROUP_DISABLED_MASK	Group Disabled State Bit Mask
0x00008000	NC_GROUP_HOMING_MASK	Group Homing State Bit Mask
0x00004000	NC_GROUP_ERROR_STOP_MASK	Group Error State Bit Mask
0x00002000	NC_GROUP_MOVING_MASK	Group Moving State Bit Mask
0x00001000	NC_GROUP_STOPPING_MASK	Group Stopping State Bit Mask

By definition, both the Axis, Vector, Accelerating (AC), Decelerating (DC), and Constant Velocity, State Bits will be set according to the sign of the velocity derivative. Therefore, in reference to Figure 5-5, the example shows the changes in state between AC, DC, and at constant velocity.

Figure 5-5 Example of Acceleration/Deceleration axis bits

To use this table, consider the following example of an error received from the G-MAS; 0x60000408. The error is broken into the following sections:

Bitwise value	Explanation of parameter
0x00000008	Axis Homing State Bit Mask
0x00000400	Axis Error Stop State Bit Mask

0x20000000	Axis/Group Error Bit Mask
0x40000000	Axis/Group Data Valid Bit Mask
0x60000408	TOTAL

The diagnostics of the error describes an Axis or group of axes data validity error causing an error stop during homing.

5.4.1. Blended Behaviour Mechanism

Within the G-MAS, function blocks are queue aligned in the execution process. Elmo has designed an execution process using the *ucExecute* Boolean variable entered to each axis group function block queue, with a specific execute flag bit. The execute bit is an input to the motion function blocks. This input defines the following:

- PreCalc - whether the recalc end velocites routine will be executed
- RealTime - whether the current function block will be executed.

Each time the execute bit "1"is entered with a function block, it causes all the previous function blocks to set their execute state to "1". For example, for five function blocks in a queue:

- Enter FB1 Execute bit "0" - no motion.
- Enter FB2 Execute bit "0" - no motion.
- Enter FB3 Execute bit "0" - no motion.
- Enter FB4 Execute bit "0" - no motion.
- Enter FB5 Execute bit "1" - perform recalc end velocities and start motion of all five function blocks.

Another example is where the following occurs in a function block queue:

- Enter FB1 Execute bit "0" - no motion.
- Enter FB2 Execute bit "0" - no motion.
- Enter FB3 Execute bit "1" - perform recalc end velocities and start motion of all three function blocks.
- Enter FB4 Execute bit "0" - no additional motion(motion limited by FB3).
- Enter FB5 Execute bit "0" - no additional motion(motion limited by FB3.
- Delay - wait until the motion of the first three function blocks is ended.
- Now, two function blocks remain in the queue (FB4, FB5), waiting, but not executed.
- Enter FB6 Execute bit "1" – to perform recalc end velocities and start motion of all three function blocks present in the queue (FB4, FB5, FB6).

Each time when the last inserted function block is inserted with execute bit "0", this causes the Pending bit to rise in the Axis state mask.

5.4.2. Special function block insertion mechanism

There are two special circumstances in the transition\arch insertion:

- Insert transition when enter to pending (pending bit is rising on current function block – execute bit 0 after a function block with execute bit 1)
- Insert transition when the last function block is active

For the Pending situation, the G-MAS allows the user to insert the second function block in blended mode with a transition, and continue the axes operation, only when the global parameter MMC_SUPPORT_BLENDED_FB_PENDING (53) is set to 1 (true). Otherwise, the operation is halted, and the second function block is inserted in MC_BUFFERED_MODE mode. The Boolean Global Parameter MMC_SUPPORT_BLENDED_FB_PENDING described in Parameters Tables

Table 1 above, is initialized to 0 and can be changed using one of the following functions, and changing the parameter to 1:

- MMC_WriteBoolParameter
- MMC_GlobalWriteBoolParameter
- MMC_GroupWriteBoolParameter

In the second situation, it is not normally possible to insert a function block with blended mode (and transition arc) when the last function block is an active function block in a axis or group. To solve this situation, allowing the axes operation to effectively continue, the BufferMode is changed to MC_BUFFERED_MODE and the TransitionMode to MC_TM_NONE_MODE in the function block to be inserted. The transition arc is not inserted. If the user requires that the function block is inserted with an arc, then insert the previous function block with execute bit 0. To obtain further information regarding the status of the function block list, invoke MMC_GetFBDepth. If the Depth value is greater than one, then the last function block is not in active execution.

5.5. Administrative Function Block Handling

This section describes the function block queue execution order. Motion function blocks are present in the "main" queue, while each queue member has a pointer to an "auxiliary" queue containing the administrative function blocks, as shown in Figure 5-6. The following Administrative Function Blocks are added to maintain the logic of this approach:

- Dwell (Precise wait).
- WriteGroupOfParameters Ability to write up to 5 parameters in a single function block call.
- Stop

In the following example, the 2nd MoveAbsolute will not be called until the two Dwells and the SetParameter functions have completed.

Figure 5-6: Motion and Administrative Function Blocks

The function blocks queue is implemented as a bi-directional linked list, and in some cases it is necessary to distinguish between motion traversing function blocks, e.g. profiler pre-calculations, and entire traversing function blocks (all), e.g. marking them as aborted. Therefore, two pairs of pointers are required, one to connect between the motion function blocks (mprev, mnext), and the other to manage the "auxiliary" administrative queue (aprev, anext). The system is then able to traverse either through the motion function block's queue, or through all function blocks, providing a generic solution for all function block type handling.

5.5.1. Classification

Generally, administrative function blocks can be divided the into two categories:

Immediate execution	The execution time for these function blocks is immediate, and the active function block can be advanced in the same cycle, e.g. SetParameter.
Non-immediate	The execution of these function blocks might take a while, and if inserted, there is no need for the profiler to traverse past the function block, e.g. Dwell.

5.5.2. Flags

Each motion function block in a queue is a potential root (head) of an administrative function blocks list. This root (head) now consists of a pre-calculation flag which changes to negative when any non-immediate function block is inserted to the current top administrative queue. This informs the profiler service functions to ignore this function block, and return NULL (for example for GetPrevFB API).

However, when an immediate execution function block is processed, the system can move to the next function block in the same clock interrupt. This requires setting the parameters which will affect the next function block, and execute the next function block immediately, in the same clock interrupt.

5.5.3. Examples

The following examples demonstrate the principle involved in using this approach to the Administrative Function Block handling.

5.5.3.1. 2D Motion with Precise Torque Change

Infinite sequence where we want to set high current to administrative axis that make a push.

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 without stopping.

5.5.3.2. IO Change at Precise Location

Infinite sequence where we want to turn light on, using digital output in a bounded specific location.

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

The motion sequence performed with blending transitions between function blocks, all the sequence without stopping.

5.5.3.3. Change Software Limits of Y axis During Motion

Infinite sequence where we want to change the SW limits of 'Y' axis during motion according to the area where we work.

① - 2D linear move.
② - 2D linear move.
③ - Update High\Low SW limits of 'Y' for "-B-".
④ - 2D linear move.
⑤ - 2D circle move.
⑥ - 2D linear move.
⑦ - Update High\Low SW limits of 'Y' for "-A-".
⑧ - 2D linear move.
⑨ - 2D linear move.
⑩ - 2D linear move.

The motion sequence performed with blending transitions between function blocks, all the sequence without stopping.

5.6. Single Axis Motion Control

Motion Control describes the motion of the single axes, either NC or Distributed according to the definitions described in section **2.2 G-MAS Operation Modes** and **2.2.3 G-MAS Axes and Node Definitions** above. The following single axis function blocks are described:

Single Axis
MMC_Halt
MMC_Home
MMC_HomeDS402
MMC_MoveAbsolute
MMC_MoveAdditive
MMC_MoveRelative
MMC_MoveVelocity
MMC_Stop
MMC_MoveAbsoluteRepetitive
MMC_MoveRelativeRepetitive
MMC_MoveAdditiveRepetitive

Since the G-MAS API operates, equally well for the SimplIQ and Gold range of motion controllers, for each of the motion axis function blocks, the motion profile must be defined whether as NC or Distributed, with the appropriate mode applicable to each possible variation.

Mode	Controller	Motion	
		NC	**Distributed**
DS-402	SimplIQ	Interpolated position mode (IP)	Depends on Profile of drive: Profile position mode (PP) Profiled velocity mode (PV) Torque profiled mode (PT) Homing mode (HM)
	Gold	Cyclic sync position mode Cyclic sync velocity mode	Depends on Profile of drive: Profile position mode (PP) Profiled velocity mode (PV) Homing mode (HM)

5.6.1. MMC_Halt

Call this function to command a controlled motion stop for a specific axis.

```
MMC_LIB_API int MMC_HaltCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_HALT_IN* pInParam,
OUT MMC_HALT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_HALT_IN** input structure that receives the HALT Information

pOutParam

Points to the **MMC_HALT_OUT** output structure receiving information as a result of calling the HALT function.

Remarks

The axis is moved to the state Discrete Motion, until the velocity is zero. With the output set to Done, the state is transferred to StandStill.

MMC_Halt is used to stop the axis under normal operation conditions.

If this command is active, the next command can be issued, e.g. a driverless vehicle detects an obstacle and needs to stop. When MMC_Halt is issued, then, before the standstill is reached the obstacle is removed and the motion can be continued by setting another motion command, so that the vehicle continues its motion and does not stop.

Scope

MMC_Halt will only operate from Standstill or Continuous Motion. It will not operate from Stopping, Homing, ErrorStop, or Disabled. Refer to the State diagram in **Figure 5-1**.

MMC_HALT_IN Structure

```
typedef struct{
    float fDeceleration;
    float fJerk;
    MC_BUFFERED_MODE_ENUM eBufferMode;
    unsigned char ucExecute;
}MMC_HALT_IN;
```

Parameters

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_HALT_OUT Structure

```
typedef struct{
    unsigned int uiHndl;
    unsigned short usStatus;
    short usErrorID;
}MMC_HALT_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-7 describes the function block for MMC_Halt

Figure 5-7: MMC_Halt function block

5.6.1.2. Function Block Code Example

```
int rc;
MMC_HALT_IN    stHalt_in;
MMC_HALT_OUT   stHalt_out;
//
// Inserting the structure parameters:
stHalt_in.fDeceleration     = 100000.0;          // Value of the acceleration
stHalt_in.fJerk             = 20000.0;           // Value of the Jerk
stHalt_in.eBufferMode       = MC_ABORTING_MODE;  // MC_BUFFERED_MODE_ENUM Defines the behavior
of the axis
stHalt_in.ucExecute         = 1;
//
rc = MMC_HaltCmd (hConn, iAxisRef, &stHalt_in, &stHalt_out);
if (rc != 0)
{
    HandleError();
}
```

5.6.1.3. Implementation Example

The example below shows the behavior of MMC_Halt in combination with MMC_MoveVelocity.

1. A rotating axis is ramped down with MMC_Halt.
2. Another motion command overrides the MMC_Halt command. MMC_Halt allows this, in contrast to MMC_Stop. The axis can accelerate again without reaching standstill.

Figure 5-8: Combination of two blocks for MMC_Halt – Example

Figure 5-9: Timing diagram of two blocks for MMC_Halt – Example

5.6.2. MMC_Home

Commands the axis to perform the Search Home sequence.

```
MMC_LIB_API int MMC_HomeCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_HOME_IN* pInParam,
OUT MMC_HOME_OUT* pOutParam
);
```

Motion Mode NC - Not Supported Distributed - Supported

Source GMAS\includes\MMC_PLCopen_single_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_HOME_IN** input structure that receives the Home command.

pOutParam

Points to the **MMC_HOME_OUT** output structure receiving information as a result of calling the HOME function.

Remarks

The details of this sequence are manufacturer dependent and can be set by the axis' parameters. The Position input is used to set the absolute position when the reference signal is detected. This function block's movement ends at StandStill.

MMC_Home is a generic function block, which performs a system specified homing procedure. This can also be constructed by the Homing function blocks described in section **Homing Functions**.

Issuing MMC_Home in any other state other than StandStill will go to ErrorStop, even if MMC_Home is issued from the state Homing itself.

Scope

MMC_Home will only operate from Standstill. It will not operate from any form of Motion, Stopping, Homing, ErrorStop, or Disabled. Refer to the State diagram in **Figure** 5-1

MMC_HOME_IN Structure

```
typedef struct{
double dbPosition;
float fAcceleration;
float fVelocity;
float fDistanceLimit;
float fTorqueLimit;
MMC_HOME_MODE_ENUM eHomingMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
MC_HOME_DIRECTION_ENUM eDirection;
MC_SWITCH_MODE_ENUM eSwitchMode;
unsigned int uiTimeLimit;
unsigned char ucExecute;
}MMC_HOME_IN;
```

Parameters

dbPosition

Target position for the motion when conditions are met. Any double -ve or +ve values in technical unit [u].

fVelocity

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s.

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2.

fDistanceLimit

Limit of the drive distance. If StepAbsSwitch condition is not met within a DistanceLimit travel, an error is issued. 0 means no distance limit. Any float +ve or –ve value in technical unit [u].

fTorqueLimit

Limit of the torque force of the drive. 0 means no torque limit. Any positive float value in Torque units

eHomingMode

MC_HomingMode enumerator type can have 1-of-5 values as described in section **Homing Functions** below:

MC_ABS_SWITCH

MC_LIMIT_SWITCH

MC_REF_PULSE

MC_DIRECT

MC_BLOCK

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Homing can only be performed from Standstill. The Aborting mode is not supported.Enumerator modes are as follows:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting	Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared
Buffered	The next function block affects the axis as soon as the previous movement is completed.
BlendingLow	The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).
BlendingPrevious	Blending with the velocity of function block 1 at the end-position of this block
BlendingNext	Blending with the velocity of function block 2 at end-position of function block1
BlendingHigh	Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

MC_HOME_DIRECTION_ENUM
eDirection

Specifies the direction of the motion if any. MC_Direction enumerator type can have 1-of-4 values depending on the MC_HomingMode parameter:

MC_POSITIVE	For MC_ABS_SWITCH, MC_BLOCK, and MC_REF_PULSE, starts in positive direction always. For MC_LIMIT_SWITCH starts in Positive direction searching positive LimitSwitch. The direction is automatically reversed from LimitSwitch initial state.
MC_NEGATIVE	For MC_ABS_SWITCH, MC_BLOCK, and MC_REF_PULSE, starts in negative direction always. For MC_LIMIT_SWITCH starts in Negative direction searching negative LimitSwitch. The direction is automatically reversed from LimitSwitch initial state.
MC_SWITCH_POSITIVE	For MC_ABS_SWITCH depends on Switch status at Execute edge. If Switch is Off, direction is positive, if On it is negative.
MC_SWITCH_NEGATIVE	For MC_ABS_SWITCH depends on Switch status at Execute edge. If Switch is On, direction is positive, if Off it is negative.

eSwitchMode

Sensor condition to finalize StepAbsSwitch in any Switch mode. MC_SwitchMode enumerator type can have 1-of-6 values:

MC_ON	When sensor is ON
MC_OFF	When sensor is OFF
MC_EDGE_ON	Rising Edge ON when Off to On transition in sensor
MC_EDGE_OFF	Rising Edge OFF when On to Off transition in sensor
MC_EDGE_SWITCH_POSITIVE	Edge depends on motion direction
MC_EDGE_SWITCH_NEGATIVE	As previous parameter but opposite

uiTimeLimit

Limit of the time for the drive to reach Home. If StepAbsSwitch condition is not met in the TimeLimit, error is issued. 0 means no time limit. Any numerical value in milli-seconds.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_HOME_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_HOME_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-10 describe the function blocks for MMC_Home.

Figure 5-10: MMC_Home function blocks

5.6.2.2. Homing Functions

The homing function supports the MC_HomingMode, as an ENUM datatype, which has the following modes:

HomeAbsSwitch	Absolute Switch homing plus limit switches
HomeLimitSwitch	Homing against limit switches
HomeBlock	Homing against hardware parts blocking movement
HomeRefPulse	Homing using encoder reference pulse Zero Mark
HomeDirect	Static Homing, forcing a position from a user reference

These modes act in the state diagram within the function block MC_Homing, are issued before an axis becomes operational, and use the state Homing during the homing command. After finalizing the homing procedure, the axis moves to the state StandStill. These modes are called active homing.

In order to describe and detail your own homing procedure, and even create user specific and customized homing procedures, function blocks are defined. Using one or more connected function blocks, you can create complex homing sequences.

The following defined function blocks match the homing procedures. They do not change the state of the axis after completion, and therefore remain in the Homing mode. However, they can easily be coupled to another homing function block. To exit the homing state, a dedicated function block has been defined: MC_FinishHoming.

Note: **In order to operate the various Homing Modes in the drive, the G-MAS will call the relevant DS-402 Homing operating mode within the drive.**

Homing Functions*	
Function Name	**Description**
MC_AbsSwitch *	Absolute Switch homing plus limit switches
MC_LimitSwitch *	Homing against limit switches
MC_Block *	Homing against hardware parts blocking movement
The following function blocks lead to a final position, automatically clearing the Homing State:	
MC_RefPulse *	Homing using encoder reference Pulse Zero Mark
MC_Direct *	Static Homing forcing a position from a user reference
In addition, homing functions are necessary while the machine is operating, i.e. not in the state Homing. This Homing-on-the-fly is called passive homing. They have no effect on the State Diagram, and similar to the administrative function blocks, can be called in any movement state. They consist of:	
MC_StepReferenceFlyingSwitch *	
MC_StepReferenceFlyingRefPulse*	
MC_AbortPassiveHoming	

5.6.2.3. Homing Procedures

A homing procedure couples a position to a specific axis. Homing is dependent on the encoders (absolute versus relative), and system used (linear versus circular). Absolute encoders do not need a movement during the homing procedure, since the exact positions can be directly transferred to the system.

For other encoder types, a movement is necessary, since there is no knowledge of the exact position within the system. This movement is at low speed in some direction until a certain measuring point is reached. Such a measuring point can be scanned from both directions to increase the precision.

Figure 5-11: Three step sequence

The above example (**Figure** 5-11) describes a simple sequence consisting of three steps:

1. Search for Signal
2. Search for another Signal
3. Move axis to a pre-defined position

5.6.2.3.a HomeAbsSwitch

Figure 5-12: Homing by search for an external switch, which is positioned absolute

This homing procedure (**Figure** 5-12) uses the MC_HomingMode function MC_ABS_SWITCH, and performs a homing function by searching for an absolute positioned external physical switch. It is called Absolute Switch because it might be active only in a specific area of the Moving Part limits, and be Off in the areas surrounding this switch area. Notice that an Absolute Switch has two Off (or On) areas, as shown in the above example. The real meaning of On and Off will depend on the switch logic and controller input logic configurations.

Additional limit switches can be applicable for the security of the system e.g. if homing started in a direction that the absolute reference cannot be found.

The physical layout has the risk that homing is started in the wrong direction (moving away from the switch). To support and correct such a situation, it implements a special behavior when Limit Switches are found (or the *AbsSwitch* itself is On at Execute):

- Axis State is set to Homing (if not already in that state).
- The homing is commanded in the most likely direction where the sensor is to be found.
- The velocity is defined by the input.
- The torque is limited.
- Both Time and Distance Limits cause an error if exceeded.
- If any LimitSwitch is found during Homing (any of them), then a special process is started in the opposite direction, the AbsSwitch is searched to switch Off (or On depending in *SwitchMode* setting). The Edge (passed by), and homing process is restarted in the original direction and with the same conditions. This ensures that the end conditions are always same.
- If the SwitchMode is either MC_SwitchNegative or MC_SwitchPositive, then the special process is also started in opposite direction depending from the switch state at Execute.
- If the *SetPosition* input is disconnected, the function block does not modify actual position. However, if the *SetPosition* input is connected, then this function block modifies the actual position to the *SetPosition* value when the homing condition is met.

Figure 5-13: HomeAbsSwitch example

The procedure itself requires limited intelligence. In the example above (**Figure** 5-13), three different starting positions (indicated by the small squares which are starting points) lead to three completely different motion sequences, but with an identical result. This behavior is possible with just one function block as shown below in **Figure** 5-14.

Figure 5-14: Example of a homing step against falling edge of sensor in negative direction

Figure 5-14 describes a homing step against a sensor falling edge in the negative direction. The actual position and axis state are not modified.

Figure 5-15: Example of a homing step against rising edge of sensor in negative direction

Figure 5-15 describes a homing step against sensor rising edge in the negative direction. The actual position and axis state are not modified.

Figure 5-16: Example of an overlapping switch configuration

Figure 5-16 describes an overlapping switch configuration, which behaves similarly to operating via the limit switches. This example also does not modify the actual position.

Figure 5-17: Additional example of a homing step against falling edge in the same negative direction

Figure 5-17 describes a homing step against the falling edge this time on the same negative direction. However, the actual position at the edge detection is modified and moved away from the homing state of the axis due to of data in SetPosition.

Figure 5-18: Example of Homing with MC_Positive direction

If the input Direction is set to a fixed direction (MC_Positive (**Figure** 5-18) or MC_Negative) then the initial switch state is ignored. This is used for example in the rotary axis where only one sense of rotation is allowed. In this situation, the axis actual position and state are not modified.

5.6.2.3.b HomeLimitSwitch

Figure 5-19: Homing by search for sensor using Limit Switches

This homing procedure uses the MC_HomingMode function MC_LIMIT_SWITCH, and performs a homing function searching for sensor using only *LimitSwitches*. A LimitSwitch has one Off (or On) area. In this procedure the following occurs:

- The axis state changes to Homing (if not already in this state).
- Home is commanded by the user in the desired homing direction at the selected Velocity.
- If a LimitSwitch is found On, on rising Execute, then the process is started in the opposite direction as specified. The search is for the LimitSwitch Off (or On, depending on LimitSwitchMode setting) Edge (released), and the process is restarted in the original direction. This ensures that the end conditions are always the same.
- The torque is limited.
- The Time and Distance Limits cause error if exceeded.
- If the SetPosition input is disconnected, this function block does not modify the actual position. However, if SetPosition input is connected, the function block modifies the actual position to the SetPosition Value when homing condition is met.

Figure 5-20: HomeLimitSwitch example

Figure 5-20 describes a situation where the result in execution originates from two different starting points (squares). The actual Position is not modified and the axis remains in homing state.

Figure 5-21: HomeLimitSwitch example with a different edge detection

Figure 5-21 describes the same conditions as in **Figure** 5-20, but enforcing different edge detection (MC_FallingEdge). The Actual Position is not modified.

5.6.2.3.c HomeBlock

Figure 5-22: Homing against a physical object

This homing procedure uses the MC_HomingMode function MC_BLOCK, and performs a homing against a physical object, mechanically blocking the movement. In this mode, there is no limit switch or Reference Pulse. Adequate torque limits are required to prevent damaging mechanics during the homing process. The Block condition is that the torque limit is reached and the actual velocity falls below the value of the fVelocity input for at least the uiTimeLimit.

The following conditions also apply:

- Axis state changes to Homing (if not already in homing).
- The homing procedure is commanded by the user in the desired homing direction at the selected Velocity.
- The Torque is limited.

- Time and Distance Limits can cause an error, if exceeded.
- Process is finished when Torque is in limit condition and real velocity is below 5% of selected velocity.
- If the SetPosition input is disconnected, this function block does not modify the actual position. However, if the SetPosition input is connected, then this function block modifies the actual position to the SetPosition Value when the condition is met.

Figure 5-23: HomeBlock example

5.6.2.3.d HomeRefPulse

Figure 5-24: Homing by search for a Zero pulse in the encoder

This homing procedure uses the MC_HomingMode function MC_REF_PULSE, and performs homing by searching for Zero pulse (also called Marker or reference pulse) in the encoder. It starts the homing step procedure at rising edge and searches for the reference pulse, which appears once per encoder revolution. The advantage in using a Reference Pulse for homing is the high accuracy and precision that can be achieved compared to traditional optical, mechanical, or magnetic sensors.

The following conditions are applicable:

- The axis state changes to Homing (if not already in that mode).
- The homing is commanded by the user in the desired homing direction at the programmed velocity.
- At the first occurrence of the reference pulse, the procedure is completed.
- Torque is limited. Time and Distance Limits can cause an error, if exceeded.
- If the SetPosition input is not connected, the function block does not modify the actual position. However, if the SetPosition input is connected, then the function block modifies the actual position to the SetPosition value when the condition is met.

Figure 5-25: HomeRefPulse example using separate function blocks

Figure 5-26: HomeRefPulse example

To perform Homing using this function it is common to perform a rough or coarse approach against a mechanical sensor at a higher velocity, and after detecting a reference pulse continue at a lower velocity. This is a traditional two-step homing using Coarse by external switch in reverse and Fine by reference pulse in forward as shown in **Figure** 5-26. To simplify, both functions could be grouped together in a single function block. The advantage of having separate function blocks (**Figure** 5-25) is that any combination is possible (MC_StepBlock and after MC_RefPulse, etc.), stating different velocity and conditions for each step, providing a higher flexibility without increasing the complexity of the homing function block.

5.6.2.3.e HomeRefPulseSet

Figure 5-27: HomeRefPulseSet by searching for a set of reference pulses

This homing procedure performs homing by searching for a set of reference pulses (also called marker or zero pulse) from the encoder. The reference pulses appear repeatedly on the distance coded linear encoder length. The advantage in using distance coded reference pulses for homing is the reduced need for traveling distance and reduced time.

A possible scenario:

1. Start in a preferred direction with low speed.
2. Find first ref pulse. If block or end switch are found, instead stop and retry the full procedure in inverse direction.
3. Find second ref pulse, if block or end switch are found instead stop and retry the full procedure in inverse direction.
4. Calculate absolute position.
5. If the procedure still finds block or end switch conditions when doing the retry, abort the process and signal a fault situation.

5.6.2.3.f HomeDistanceCoded

Figure 5-28: Schematic representation of an incremental graduation with distance-coded reference marks

With the incremental measuring method, the graduation consists of a periodic grating structure. The position information is obtained by counting the individual increments (measuring steps) from some point of origin. Since an absolute reference is required to ascertain positions, the scales or scale tapes are provided with an additional track that bears a reference mark. The absolute position on the scale, established by the reference mark, is gated with exactly one signal period. The reference mark must therefore be scanned to establish an absolute reference or to find the last selected datum.

In some cases, this may necessitate machine movement over large lengths of the measuring range. To simplify such Reference Runs, many encoders feature distance-coded reference marks—multiple reference marks that are individually spaced according to a mathematical algorithm. The subsequent electronics find the absolute reference after traversing two successive reference marks—only a few millimeters traverse (**Figure** 5-28).

With distance-coded reference marks, the absolute reference is calculated by counting the signal periods between two reference marks. After calculation, it sets the actual position. If there are no coded marks found within a certain time or distance traveled, it generates an error.

5.6.2.3.g HomeDirect

This homing procedure uses the MC_HomingMode function MC_DIRECT, and performs a static homing by directly forcing an actual position to the SetPosition input value. No physical motion is performed in this mode. This is equivalent to a MC_SetPosition action, but clears the Homing State.

5.6.2.3.h HomeAbsolute

This homing procedure performs a static homing function by setting the actual position to the position of an absolute encoder.

No physical motion is performed in this mode. It is equivalent to issuing MC_SetPosition with SetPosition obtained from the absolute encoder reading, but after finalizing the Homing State.

5.6.2.3.i MC_FinishHoming

This function block transfers an axis from the state Homing to the state StandStill and finalizes the homing procedure. In addition, it can perform a relative movement to move the axis in the allowed area (working area), in a situation whereby the homing steps left the axis on the wrong side of the switch.

5.6.2.4. Function Block Code Example

```
int rc;
MMC_HOME_IN   stHome_in;
MMC_HOME_OUT  stHome_out;
//
// Inserting the structure parameters:
stHome_in.fAcceleration  = 100000.0;             // Value of the acceleration
stHome_in.fDistanceLimit = 100000.0;             // Limit of the drive distance
stHome_in.fTorqueLimit   = 2.0;                  // Limit of the torque force of the drive
stHome_in.eHomingMode    = MC_DIRECT;            // MC_HomingMode enumerator type
stHome_in.eDirection     = MC_POSITIVE_DIRECTION;// MC_Direction Enumerator type
stHome_in.eBufferMode    = MC_BUFFERED_MODE;     // MC_BUFFERED_MODE_ENUM Defines the behavior
of the axis
stHome_in.eSwitchMode    = MC_ON; // Sensor condition to finalize StepAbsSwitch in any Switch
mode
stHome_in.dbPosition     = 100000.0;             // Target position for the motion
stHome_in.fVelocity      = 5000.0;               // Velocity in
stHome_in.uiTimeLimit    = 100;                 // Limit of the time for the drive to reach
Home
stHome_in.ucExecute      = 1;
//
rc = MMC_HomeCmd (hConn, iAxisRef, &stHome_in, &stHome_out);
if (rc != 0)
{
    HandleError();
}
```

5.6.2.5. Implementation Example

TBD

5.6.3. MMC_HomeDS402

Commands the axis to perform the Search Home sequence and can be set by the axes parameters.

```
MMC_LIB_API int MMC_HomeDS402Cmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_HOMEDS402_IN* pInParam,
OUT MMC_HOME_OUT* pOutParam
);
```

Motion Mode	NC - Not Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_HOMEDS402_IN** input structure that receives the Home DS-402 command.

pOutParam

Points to the **MMC_HOME_OUT** output structure receiving information as a result of calling the Home DS-402 function.

Remarks

The Position input is used to set the absolute position when the reference signal is detected. This function block's movement ends at StandStill.

MMC_HomeDS402 is a generic function block, which performs a system specified homing procedure. This can also be constructed by the Homing methods described in section 9.4 in the manual www.elmomc.com/support/manuals/MAN-CAN402IG.pdf.

Scope

MMC_HomeDS402 will only operate from Standstill. It will not operate from any form of Motion, Stopping, Homing, ErrorStop, or Disabled. Refer to the State diagram in **Figure 5-1**.

Issuing MMC_HomeDS402 in any other state other than StandStill will go to ErrorStop, even if MMC_HomeDS402 is issued from the state Homing itself.

MMC_HOMEDS402_IN Structure

```
typedef struct{
double dbPosition;
float fAcceleration;
float fVelocity;
float fDistanceLimit;
float fTorqueLimit;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned int uiHomingMethod;
unsigned int uiTimeLimit;
unsigned char ucExecute;
}MMC_HOMEDS402_IN;
```

Parameters

dbPosition

The position parameter is actually an offset of the position. When the homing procedure has completed, the absolute position set is –Position. Therefore, if the ID position is 500, the position at the end of the homing procedure will be -500. Any -ve or +ve double values in technical unit [u].

fVelocity

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2

fDistanceLimit

Limit of the drive distance. if StepAbsSwitch condition is not met within a DistanceLimit travel, an error is issued. 0 means no distance limit. Any +ve or –ve float value in technical unit [u]

fTorqueLimit

Limit of the torque force of the drive. 0 means no torque limit. Any positive float value in Torque units

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Homing can only be performed from Standstill. The Aborting mode is not supported.Enumerator modes are as follows:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

uiHomingMethod

DS-402 standard for homing of drive. Integer of value 1 – 36, with 4 values reserved. Refer to the CiA definition or Drive reference manual.

uiTimeLimit

Limit of the time for the drive to reach Home. If StepAbsSwitch condition is not met in the TimeLimit, error is issued. 0 means no time limit. Any numerical value in milli-seconds.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). TRUE/FALSE Boolean value.

MMC_HOME_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_HOME_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block:

- MC_TimeLimitExceeded
- MC_DistanceLimitExceeded
- MC_TorqueLimitExceeded

Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-29 describe the function blocks for MMC_HomeDS402.

Figure 5-29: MMC_HomeDS402 function blocks

5.6.3.2. Function Block Code Example

```
int rc ;
MMC_HOMEDS402_IN  stHomeDS402_in;
MMC_HOME_OUT      stHome_out;
//
// Inserting the structure parameters:
stHomeDS402_in.fAcceleration  = 100000.0;              // Value of the acceleration
stHomeDS402_in.fDistanceLimit = 100000.0;              // Limit of the drive distance
stHomeDS402_in.fTorqueLimit   = 2.0;                   // Limit of the torque force of the
drive
stHomeDS402_in.eBufferMode    = MC_ABORTING_MODE;      // MC_BUFFERED_MODE_ENUM Defines the
behavior of the axis
stHomeDS402_in.uiHomingMethod = 1;                     // DS-402 standard for homing of drive
stHomeDS402_in.dbPosition     = 100000.0;              // Target position for the motion
stHomeDS402_in.fVelocity      = 5000.0;                // Velocity in
stHomeDS402_in.uiTimeLimit    = 100000;               // Limit of the time for the drive to
reach Home
stHomeDS402_in.ucExecute      = 1;
//
rc = MMC_HomeDS402Cmd (hConn, iAxisRef, &stHomeDS402_in, &stHome_out) ;
if (rc != 0)
{
    HandleError() ;
}
```

5.6.3.3. Implementation Example

5.6.4. MMC_MoveAbsolute

Commands a discreet controlled motion for a single axis to a specified absolute position.

```
MMC_LIB_API int MMC_MoveAbsoluteCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVEABSOLUTE_IN* pInParam,
OUT MMC_MOVEABSOLUTE_OUT* pOutParam
);
```

Motion Mode	NC - All Buffering modes are supported.	Distributed - Only MC_ABORTING_MODE Buffered mode is supported.
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVEABSOLUTE_IN** input data structure using the MMC_MoveAbsolute function.

pOutParam

Points to the **MMC_MOVEABSOLUTE_OUT** output structure receiving information as a result of calling the MMC_MoveAbsolute function.

Remarks

This action completes with velocity zero if no further action is pending.

Scope

MMC_MoveAbsolute will only operate from Standstill or Continuous Motion. It will not operate from Stopping, Homing, ErrorStop, or Disabled. Refer to the State diagram in **Figure** 5-1.

MMC_MOVEABSOLUTE_IN Structure

```
typedef struct{
double dbPosition;
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_DIRECTION_ENUM eDirection;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
}MMC_MOVEABSOLUTE_IN;
```

Parameters

dbPosition

Target position for the motion when conditions are met. Any -ve or +ve double values in technical unit [u].

fVelocity

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eDirection

Specifies the direction of the motion, if any. The MC_DIRECTION_ENUM enumerator type can have 1-of-4 values:

MC_NONE_DIRECTION = 0

MC_POSITIVE_DIRECTION = 1

MC_SHORTEST_WAY = 2 Not implemented as yet

MC_NEGATIVE_DIRECTION = 3

MC_CURRENT_DIRECTION = 4

The Enum type MC_SHORTEST_WAY is focused to a trajectory, which will go through the shortest route (option not implemented as yet). The decision which direction to move is based on the current position from where the command is issued. In general,

make sure to set the parameter to MC_POSITIVE_DIRECTION to be sent.

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVEABSOLUTE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVEABSOLUTE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-30 describes the function block for MMC_MoveAbsolute

Figure 5-30: MMC_MoveAbsolute function block

5.6.4.2. Function Block Code Example

```
int rc;
MMC_MOVEABSOLUTE_IN    stMove_Abs_in;
MMC_MOVEABSOLUTE_OUT   stMove_Abs_out;
//
// Inserting the structure parameters:
stMove_Abs_in.fAcceleration = 100000.0;              // Value of the acceleration
stMove_Abs_in.fDeceleration = 100000.0;              // Value of the deceleration
stMove_Abs_in.fJerk         = 20000000.0;                // Value of the Jerk
stMove_Abs_in.eDirection    = MC_POSITIVE_DIRECTION; // MC_Direction Enumerator type
stMove_Abs_in.eBufferMode   = MC_BUFFERED_MODE;      // MC_BUFFERED_MODE_ENUM Defines the
behavior of the axis
stMove_Abs_in.dbPosition    = 100000.0;              // Target position for the motion
stMove_Abs_in.fVelocity     = 5000.0;                // Velocity in
stMove_Abs_in.ucExecute     = 1;
//
rc = MMC_MoveAbsoluteCmd (hConn, iAxisRef, &stMove_Abs_in, &stMove_Abs_out);
if (rc != 0)
{
    HandleError();
}
```

5.6.4.3. Implementation Example

The following figure shows two examples of the combination of two absolute move function blocks:

1. The left part of timing diagram in **Figure** 5-32 illustrates the case where the second function block is called after the first one. If the first block reaches the commanded position of 6000 (and the velocity is 0) then the output Done causes the Second function block to move to the position 10000.
2. The right part of the timing diagram illustrates the case if the second move function Block starts the execution while the first function block is still executing. In this case the first motion is interrupted and aborted by the Test signal during the constant velocity of the First function block. The second function block moves directly to the position 10000 although the position of 6000 is not yet reached.

Figure 5-31: Combination of two blocks moved absolute - Example

Figure 5-32: Timing diagram for MMC_MoveAbsolute – Example

5.6.5. MMC_MoveAdditive

Commands a controlled motion of a specified relative distance additional to the most recent commanded position in the discrete motion state.

```
MMC_LIB_API int MMC_MoveAdditiveCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVEADDITIVE_IN* pInParam,
OUT MMC_MOVEADDITIVE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVEADDITIVE_IN** input data structure using the MMC_MoveAdditive function.

pOutParam

Points to the **MMC_MOVEADDITIVE_OUT** output structure receiving information as a result of calling the MMC_MoveAdditive function.

Remarks

The most recent commanded position may be the result of a previous MMC_MoveAdditive motion, which was aborted. If the function block is activated in the Continuous Mode, the specified relative distance is added to the actual position at the time of the execution.

Scope

MMC_MoveAdditive will only operate from Standstill, Continuous Motion, or Discrete Motion. It will not operate from Stopping, Homing, ErrorStop, or Disabled. Refer to the State diagram in **Figure** 5-1.

MMC_MOVEADDITIVE_IN Structure

```
typedef struct{
double dbDistance;
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_DIRECTION_ENUM eDirection;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
}MMC_MOVEADDITIVE_IN;
```

Parameters

dbDistance

Relative additive distance for the motion. Any -ve or +ve double values in technical unit [u].

fVelocity

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eDirection

Specifies the direction of the motion, if any. The MC_DIRECTION_ENUM enumerator type can have 1-of-4 values:

MC_NONE_DIRECTION = 0

MC_POSITIVE_DIRECTION = 1

MC_SHORTEST_WAY = 2 Not implemented as yet

MC_NEGATIVE_DIRECTION = 3

MC_CURRENT_DIRECTION = 4

The Enum type MC_SHORTEST_WAY is focused to a trajectory, which will go through the shortest route(option not implemented as yet). The decision which direction to move is based on the current position from where the command is issued. In general, make sure to set the parameter to MC_POSITIVE_DIRECTION to be sent.

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVEADDITIVE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVEADDITIVE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-33 describes the function block for MMC_MoveAdditive

Figure 5-33: MMC_MoveAdditive function block

5.6.5.2. Function Block Code Example

```
int rc ;
MMC_MOVEADDITIVE_IN    stMoveadd_in ;
MMC_MOVEADDITIVE_OUT   stMoveadd_out ;
//
// Inserting the structure parameters:
stMoveadd_in.fAcceleration = 100000.0;              // Value of the acceleration
stMoveadd_in.fDeceleration = 100000.0;              // Value of the deceleration
stMoveadd_in.fJerk         = 20000000.0;            // Value of the Jerk
stMoveadd_in.eDirection    = MC_POSITIVE_DIRECTION; // MC_Direction Enumerator type
stMoveadd_in.eBufferMode   = MC_BUFFERED_MODE;      // MC_BUFFERED_MODE_ENUM Defines the
behavior of the axis
stMoveadd_in.dbDistance    = 100000.0;              // Relative distance for the motion
stMoveadd_in.fVelocity     = 5000.0;                // Velocity in
stMoveadd_in.ucExecute     = 1;
//
rc = MMC_MoveAdditiveCmd (hConn, iAxisRef, &stMoveadd_in, &stMoveadd_out) ;
if (rc != 0)
{
    HandleError() ;
}
```

5.6.5.3. Implementation Example

The following figure shows the example of the combination of move absolute and additive function blocks.

1. The left part of timing diagram illustrates the case if the Second function block is called after the First one. If First reaches the commanded distance 6000 (and the velocity is 0) then the output Done causes the Second function block to move to the distance 10000.
2. The right part of the timing diagram illustrates the case if the Second move function block starts the execution while the First function block is still executing. In this case the First motion is interrupted and aborted by the Test signal during the constant velocity of the First function block. The Second function block adds on the previous commanded position of 6000 the distance 4000 and moves the axis to the resulting position of 10000.

Figure 5-34: Combination of two blocks for MMC_MoveAdditive with Timing diagram – Example

5.6.6. MMC_MoveRelative

Commands a discreet controlled motion of a specified distance relative to the set position at the time of the execution.

```
int MMC_MoveRelativeCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVERELATIVE_IN* pInParam,
OUT MMC_MOVERELATIVE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVERELATIVE_IN** input data structure using the MMC_MoveRelative function.

pOutParam

Points to the **MMC_MOVERELATIVE_OUT** output structure receiving information, as a result of calling the MMC_MoveRelative function.

Remarks

This action completes with velocity zero if no further action are pending.

Scope

MMC_MoveRelative will only operate from Standstill or Continuous Motion. It will not operate from Stopping, Homing, ErrorStop, or Disabled. Refer to the State diagram in **Figure** 5-1.

MMC_MOVERELATIVE_IN Structure

```
typedef struct{
double dbDistance;
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_DIRECTION_ENUM eDirection;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
}MMC_MOVERELATIVE_IN;
```

Parameters

dbDistance

Relative additive distance for the motion. Any -ve or +ve double values in technical unit [u].

fVelocity

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eDirection

Specifies the direction of the motion, if any. The MC_DIRECTION_ENUM enumerator type can have 1-of-4 values:

MC_NONE_DIRECTION = 0

MC_POSITIVE_DIRECTION = 1

MC_SHORTEST_WAY = 2 Not implemented as yet

MC_NEGATIVE_DIRECTION = 3

MC_CURRENT_DIRECTION = 4

The Enum type MC_SHORTEST_WAY is focused to a trajectory, which will go through the shortest route(option not implemented as yet). The decision which direction to move is based on the current position from where the command is issued. In general, make sure to set the parameter to MC_POSITIVE_DIRECTION to be sent.

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting	Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared
Buffered	The next function block affects the axis as soon as the previous movement is completed.
BlendingLow	The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).
BlendingPrevious	Blending with the velocity of function block 1 at the end-position of this block
BlendingNext	Blending with the velocity of function block 2 at end-position of function block1
BlendingHigh	Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVERELATIVE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVERELATIVE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-35 describes the function block for MMC_MoveRelative

Figure 5-35: MMC_MoveRelative function block

5.6.6.2. Function Block Code Example

```
int rc;
MMC_MOVERELATIVE_IN    stMoveReltv_in;
MMC_MOVERELATIVE_OUT   stMoveReltv_out;
//
// Inserting the structure parameters:
stMoveReltv_in.fAcceleration  = 100000.0;              // Value of the acceleration
stMoveReltv_in.fDeceleration  = 100000.0;              // Value of the deceleration
stMoveReltv_in.fJerk          = 20000000.0;            // Value of the Jerk
stMoveReltv_in.eBufferMode    = MC_BUFFERED_MODE;// MC_BUFFERED_MODE_ENUM Defines the behavior
of the axis
stMoveReltv_in.eDirection     = MC_POSITIVE_DIRECTION; // MC_Direction Enumerator type
stMoveReltv_in.dbDistance     = 100000.0;              // Relative distance for the motion
stMoveReltv_in.fVelocity      = 5000.0;                // Velocity in
stMoveReltv_in.ucExecute      = 1;
//
rc = MMC_MoveRelativeCmd (hConn, iAxisRef, &stMoveReltv_in, &stMoveReltv_out);
if (rc != 0)
{
    HandleError();
}
```

5.6.6.3. Implementation Example

The following figure shows the example of the combination of two relative move function blocks.

1. The left part of timing diagram illustrates the case if the second function block is called after the First one. If First reaches the commanded distance 6000 (and the velocity is 0) then the output Done causes the Second function block to move to the distance 10000.
2. The right part of the timing diagram illustrates the case if the Second move function blocks starts the execution while the First function block is still executing. In this case the First motion is interrupted and aborted by the Test signal during the constant velocity of the First function block. The Second function block adds on the actual position of 3250 the distance 4000 and moves the axis to the resulting position of 7250.

Figure 5-36: Example of two MoveRelative function blocks

Figure 5-37: Combination of two blocks for MMC_MoveRelative with Timing diagram – Example

5.6.7. MMC_MoveVelocity

Commands a continuous controlled motion at a specified velocity.

```
MMC_LIB_API int MMC_MoveVelocityCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVEVELOCITY_IN* pInParam,
OUT MMC_MOVEVELOCITY_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVEVELOCITY_IN** input data structure using the MMC_MoveVelocity function.

pOutParam

Points to the **MMC_MOVEVELOCITY_OUT** output structure receiving information as a result of calling the MMC_MoveVelocity function.

Remarks

To stop the motion, the function block has to be interrupted by another function block issuing a new command.

The signal InVelocity has to be reset when the block is aborted by another block.

Scope

MMC_MoveVelocity will only operate from Standstill or Discrete Motion. It will not operate from Stopping, Homing, ErrorStop, or Disabled. Refer to the State diagram in **Figure** 5-1. Therefore, MMC_MoveVelocity should only be used with the MC_BUFFERED_MODE_ENUM eBufferMode enumerator value of MC_ABORTING_MODE.

MMC_MOVEVELOCITY_IN Structure

```
typedef struct{
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_DIRECTION_ENUM eDirection;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
}MMC_MOVEVELOCITY_IN;
```

Parameters

fVelocity

Value of the maximum velocity (not necessarily reached). The velocity states the direction and can be a positive or negative value in u/s.

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

MC_DIRECTION_ENUM
eDirection

Specifies the direction of the motion, if any. MC_Direction enumerator type can have 1-of-4 values:

MC_NONE_DIRECTION = 0

MC_POSITIVE_DIRECTION = 1

MC_SHORTEST_WAY = 2 Not implemented as yet

MC_NEGATIVE_DIRECTION = 3

MC_CURRENT_DIRECTION = 4

The Enum type MC_SHORTEST_WAY is focused to a trajectory, which will go through the shortest route(option not implemented as yet). The decision which direction to move is based on the current position from where the command is issued.

In general in the interim, make sure to set the parameter to MC_POSITIVE_DIRECTION. In the future, all above parameters will be available.

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting	Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared
Buffered	The next function block affects the axis as soon as the previous movement is completed.
BlendingLow	The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).
BlendingPrevious	Blending with the velocity of function block 1 at the end-position of this block
BlendingNext	Blending with the velocity of function block 2 at end-position of function block1
BlendingHigh	Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVEVELOCITY_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVEVELOCITY_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-38 describes the function block for MMC_MoveVelocity

Figure 5-38: MMC_MoveVelocity function block

5.6.7.2. Function Block Code Example

```
int rc;
MMC_MOVEVELOCITY_IN    stMoveVel_in;
MMC_MOVEVELOCITY_OUT   stMoveVel_out;
//
// Inserting the structure parameters:
stMoveVel_in.fAcceleration  = 100000.0;              // Value of the acceleration
stMoveVel_in.fDeceleration  = 100000.0;              // Value of the deceleration
stMoveVel_in.fJerk          = 20000000.0;            // Value of the Jerk
stMoveVel_in.eBufferMode    = MC_BUFFERED_MODE;      // MC_BUFFERED_MODE_ENUM Defines the
behavior of the axis
stMoveVel_in.eDirection     = MC_POSITIVE_DIRECTION; // MC_Direction Enumerator type
stMoveVel_in.fVelocity      = 5000.0;                // Velocity in
stMoveVel_in.ucExecute      = 1;
//
rc = MMC_MoveVelocityCmd (hConn, iAxisRef, &stMoveVel_in, &stMoveVel_out);
if (rc != 0)
{
    HandleError();
}
```

5.6.7.3. Implementation Example

The following figure shows two examples of the combination of two *MoveVelocity* function blocks:

1. The left part of timing diagram illustrates the case if the Second function block is called after the First one is completed. If First reaches the commanded velocity 3000 then the output First.InVelocity AND the signal Next causes the Second function block to move to the velocity 2000. In the next cycle First.InVelocity is Reset and First.commandAborted is SET. Therefore, the Execute of the 2nd function block is Reset. In addition, as soon as the axis reaches Velocity 2000 the Second.InVelocity is set for 1 cycle (because the Second.Execute is set for only 1 cycle).
2. The right part of the timing diagram illustrates the case if the Second move Function Block starts the execution while the First function block is not yet InVelocity.
   The following sequence is shown:
   a. The First motion is started again by Go at the input First.Execute.
   b. While the First function block is still accelerating to reach the velocity 3000 the First function block will be interrupted and aborted because the Test signal starts the Run of the Second function block.
   c. Now the Second function block runs and decelerates the velocity to 2000.

Figure 5-39: Combination of two blocks for MC_MoveVelocity with Timing diagram – Example

Note: **In the above example, the 2nd function block in mode Aborting (If in buffered mode the velocity would reach 3000 before actuating the next function block).**

5.6.8. MMC_MoveAbsoluteRepetitive

This function receives as one of the input arguments, the command to move to the absolute target position. The axis moves between the current and target position until interrupted by any allowed function block in Aborting mode.

```
MMC_LIB_API int MMC_MoveAbsoluteRepetitiveCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVEABSOLUTEREPETITIVE_IN* pInParam,
OUT MMC_MOVEABSOLUTEREPETITIVE_OUT* pOutParam
);
```

Motion Mode	NC - All Buffering modes are supported.	Distributed - Not supported.
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVEABSOLUTEREPETITIVE_IN** input data structure using the MMC_MoveAbsoluteRepetitive function.

pOutParam

Points to the **MMC_MOVEABSOLUTEREPETITIVE_OUT** output structure receiving information, as a result of calling the MMC_MoveAbsoluteRepetitive function.

Remarks

It should be noted that all repetitive mode motion in iterpolated opMode will not operate in Abort Mode. An error is produced (GMAS Error ID -8).

Scope

The function block can only be called in StandStill mode. Calling MMC_Stop, MMC_Halt in aborting mode will stop the motion.

MMC_MOVEABSOLUTEREPETITIVE_IN Structure

```
typedef struct{
double dbPosition;
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_DIRECTION_ENUM eDirection;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned int uiExecDelayMs;
unsigned char ucExecute;
}MMC_MOVEABSOLUTEREPETITIVE_IN;
```

Parameters

dbPosition

Target position for the motion when conditions are met. Any -ve or +ve double values in technical unit [u].

fVelocity

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eDirection

Specifies the direction of the motion, if any. The MC_DIRECTION_ENUM enumerator type can have 1-of-4 values:

MC_NONE_DIRECTION = 0

MC_POSITIVE_DIRECTION = 1

MC_SHORTEST_WAY = 2 Not implemented as yet

MC_NEGATIVE_DIRECTION = 3

MC_CURRENT_DIRECTION = 4

The Enum type MC_SHORTEST_WAY is focused to a trajectory, which will go through the shortest route(option not implemented as yet). The decision which direction to move is based on the current position from where the command is issued. In general, make sure to set the parameter to MC_POSITIVE_DIRECTION to be sent.

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows:

MC_ABORTING_MODE = 1

MC_BUFFERED_MODE = 2

MC_BLENDING_LOW_MODE = 3

MC_BLENDING_PREVIOUS_MODE = 4

MC_BLENDING_NEXT_MODE = 5

MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

uiExecDelayMs

The delay in execution of the next action (in msecs). Any +ve integer value.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVEABSOLUTEREPETITIVE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVEABSOLUTEREPETITIVE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-40 describes the function block for MMC_MoveAbsoluteRepetitive

Figure 5-40: MMC_MoveAbsoluteRepetitive function block

5.6.8.2. Function Block Code Example

```
int rc;
MMC_MOVEABSOLUTEREPETITIVE_IN   stMoveAbsRepIn;
MMC_MOVEABSOLUTEREPETITIVE_OUT  stMoveAbsRepOut;
//
stMoveAbsRepIn.dbPosition       = 100000.0;
stMoveAbsRepIn.eBufferMode      = MC_BUFFERED_MODE_ENUM;
stMoveAbsRepIn.eDirection       = MC_POSITIVE_DIRECTION;
stMoveAbsRepIn.fAcceleration    = 1000000.0;
stMoveAbsRepIn.fDeceleration    = 1000000.0;
stMoveAbsRepIn.fJerk            = 10000000.0;
stMoveAbsRepIn.fVelocity        = 100000.0;
stMoveAbsRepIn.ucExecute        = 1;
stMoveAbsRepIn.uiExecDelayMs    = 0;
//
rc = MMC_MoveAbsoluteRepetitiveCmd (hConn, iAxisRef, &stMoveAbsRepIn, &stMoveAbsRepOut);
if (rc != 0)
{
HandleError();
}
```

5.6.8.3. Implementation Example

5.6.9. MMC_MoveRelativeRepetitive

This function receives as one of the input arguments, the command to move to a distance relative to the current position. The axis moves between the current and target position until interrupted by any allowed function block in Aborting mode.

```
MMC_LIB_API int MMC_MoveRelativeRepetitiveCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVERELATIVEREPETITIVE_IN* pInParam,
OUT MMC_MOVERELATIVEREPETITIVE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed – Not supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVERELATIVE_IN** input data structure using the MMC_MoveRelativeRepetitive function.

pOutParam

Points to the **MMC_MOVERELATIVE_OUT** output structure receiving information, as a result of calling the MMC_MoveRelativeRepetitive function.

Remarks

It should be noted that all repetitive mode motion will not operate in Abort Mode.

Scope

The function block can only be called in StandStill mode. Calling MMC_Stop, MMC_Halt in aborting mode will stop the motion.

MMC_MOVERELATIVEREPETITIVE_IN Structure

```
typedef struct{
double dbDistance;
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_DIRECTION_ENUM eDirection;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned int uiExecDelayMs;
unsigned char ucExecute;
}MMC_MOVERELATIVEREPETITIVE_IN;
```

Parameters

dbDistance

Relative additive distance for the motion. Any -ve or +ve double values in technical unit [u].

fVelocity

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eDirection

Specifies the direction of the motion, if any. The MC_DIRECTION_ENUM enumerator type can have 1-of-4 values:

MC_NONE_DIRECTION = 0

MC_POSITIVE_DIRECTION = 1

MC_SHORTEST_WAY = 2 Not implemented as yet

MC_NEGATIVE_DIRECTION = 3

MC_CURRENT_DIRECTION = 4

The Enum type MC_SHORTEST_WAY is focused to a trajectory, which will go through the shortest route (option not implemented as yet). The decision which direction to move is based on the current position from where the command is issued. In general,

make sure to set the parameter to MC_POSITIVE_DIRECTION to be sent.

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

uiExecDelayMs

The delay in execution of the next action (in msecs). Any +ve integer value.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVERELATIVEREPETITIVE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVERELATIVEREPETITIVE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-41 describes the function block for MMC_MoveRelativeRepetitive

Figure 5-41: MMC_MoveRelativeRepetitive function block

5.6.9.2. Function Block Code Example

```
int rc;
MMC_MOVERELATIVEREPETITIVE_IN   stMoveRelRepIn;
MMC_MOVERELATIVEREPETITIVE_OUT  stMoveRelRepOut;
//
stMoveRelRepIn.dbDistance       = 100000.0;
stMoveRelRepIn.eBufferMode      = MC_BUFFERED_MODE_ENUM;
stMoveRelRepIn.eDirection       = MC_POSITIVE_DIRECTION;
stMoveRelRepIn.fAcceleration    = 1000000.0;
stMoveRelRepIn.fDeceleration    = 1000000.0;
stMoveRelRepIn.fJerk            = 10000000.0;
stMoveRelRepIn.fVelocity        = 100000.0;
stMoveRelRepIn.ucExecute        = 1;
stMoveRelRepIn.uiExecDelayMs    = 0;
//
rc = MMC_MoveRelativeRepetitiveCmd (hConn, iAxisRef, &stMoveRelRepIn, &stMoveRelRepOut);
if (rc != 0)
{
HandleError();
}
```

5.6.10. MMC_MoveAdditiveRepetitive

This function receives as one of the input arguments, the command to move to a distance relative to the final position of the last command. The axis moves between the current and target position until interrupted by any allowed function block in Aborting mode.

```
MMC_LIB_API int MMC_MoveAdditiveRepetitiveCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVEADDITIVEREPETITIVE_IN* pInParam,
OUT MMC_MOVEADDITIVEREPETITIVE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed – Not supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVEADDITIVEREPETITIVE_IN** input data structure using the MMC_MoveAdditiveRepetitive function.

pOutParam

Points to the **MMC_MOVEADDITIVEREPETITIVE_OUT** output structure receiving information, as a result of calling the MMC_MoveAdditiveRepetitive function.

Remarks

It should be noted that all repetitive mode motion will not operate in Abort Mode.

Scope

The function block can only be called in StandStill mode. Calling MMC_Stop, MMC_Halt in aborting mode will stop the motion.

MMC_MOVEADDITIVEREPETITIVE_IN Structure

```
typedef struct{
double dbDistance;
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_DIRECTION_ENUM eDirection;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned int uiExecDelayMs;
unsigned char ucExecute;
}MMC_MOVEADDITIVEREPETITIVE_IN;
```

Parameters

dbDistance

Relative additive distance for the motion. Any -ve or +ve double values in technical unit [u].

fVelocity

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eDirection

Specifies the direction of the motion, if any. The MC_DIRECTION_ENUM enumerator type can have 1-of-4 values:

MC_NONE_DIRECTION = 0

MC_POSITIVE_DIRECTION = 1

MC_SHORTEST_WAY = 2 Not implemented as yet

MC_NEGATIVE_DIRECTION = 3

MC_CURRENT_DIRECTION = 4

The Enum type MC_SHORTEST_WAY is focused to a trajectory, which will go through the shortest route(option not implemented as yet). The decision which direction to move is based on the current position from where the command is issued. In general,

make sure to set the parameter to MC_POSITIVE_DIRECTION to be sent.

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

uiExecDelayMs

The delay in execution of the next action (in msecs). Any +ve integer value.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVEADDITIVEREPETITIVE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVEADDITIVEREPETITIVE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-42 describes the function block for MMC_MoveAdditiveRepetitive

Figure 5-42: MMC_MoveAdditiveRepetitive function block

5.6.10.2. Function Block Code Example

```
int rc;
MMC_MOVEADDITIVEREPETITIVE_IN   stMoveAddRepIn;
MMC_MOVEADDITIVEREPETITIVE_OUT  stMoveAddRepOut;
//
stMoveAddRepIn.dbPosition       = 100000.0;
stMoveAddRepIn.eBufferMode      = MC_BUFFERED_MODE_ENUM;
stMoveAddRepIn.eDirection       = MC_POSITIVE_DIRECTION;
stMoveAddRepIn.fAcceleration    = 1000000.0;
stMoveAddRepIn.fDeceleration    = 1000000.0;
stMoveAddRepIn.fJerk            = 10000000.0;
stMoveAddRepIn.fVelocity        = 100000.0;
stMoveAddRepIn.ucExecute        = 1;
stMoveAddRepIn.uiExecDelayMs    = 0;
//
rc = MMC_MoveAdditiveRepetitiveCmd (hConn, iAxisRef, &stMoveAddRepIn, &stMoveAddRepOut);
if (rc != 0)
{
    HandleError();
}
```

5.6.10.3. Implementation Example

5.6.11. MMC_Stop

Commands a controlled motion stop and transfers the axis to the state Stopping.

```
MMC_LIB_API int MMC_StopCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_STOP_IN* pInParam,
OUT MMC_STOP_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_STOP_IN** input data structure using the MMC_Stop function.

pOutParam

Points to the **MMC_STOP_OUT** output structure receiving information as a result of calling the MMC_Stop function.

Remarks

It aborts any ongoing function block execution. While the axis is in state Stopping, no other function block can perform any motion on the same axis. After the axis has reached velocity zero, the Done output is set to TRUE immediately. The axis remains in the state Stopping as long as Execute is still TRUE or velocity zero is not yet reached. As soon as Done is SET and Execute is FALSE the axis goes to state StandStill.
The state Stopping remains operational as long as the input Execute is true. The Done output is set when the stop ramp is completed. The following should be noted:

- This function block is primarily intended for emergency stop functions or exception situations
- As long as Execute is high, the axis remains in the state Stopping and may not execute any other command.
- If Deceleration = 0, the behavior of the function block is implementation specific.

Scope

MMC_Stop will only operate from Homing, Standstill or all Motion modes. It will not operate from ErrorStop, or Disabled. Refer to the State diagram in **Figure 5-1**. Calling the function block MMC_Stop in the state StandStill changes the state to Stopping and back to Standstill when Execute = FALSE.

MMC_STOP_IN Structure

```
typedef struct{
float fDeceleration;
float fJerk;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
}MMC_STOP_IN;
```

Parameters

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_STOP_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_STOP_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-43 describes the function block for MMC_Stop

Figure 5-43: MMC_Stop function block

Figure 5-44: MMC_Stop timing diagram

5.6.11.2. Function Block Code Example

```
int rc;
MMC_STOP_IN    stStop_in;
MMC_STOP_OUT   stStop_out;
//
// Inserting the structure parameters:
stStop_in.fDeceleration     = 1000000.0;          // Value of the acceleration
stStop_in.fJerk             = 2000000.0;          // Value of the Jerk
stStop_in.eBufferMode       = MC_ABORTING_MODE;   // MC_BUFFERED_MODE_ENUM Defines the behavior
of the axis
stStop_in.ucExecute         = 1;
//
rc = MMC_StopCmd (hConn, iAxisRef, &stStop_in, &stStop_out);
if (rc != 0)
{
    HandleError();
}
```

5.6.11.3. Implementation Example

The example below shows the behavior of MMC_Stop in combination with an MMC_MoveVelocity.

1. A rotating axis is ramped down with function block MMC_Stop.
2. The axis rejects motion commands as long as MC_Stop parameter Execute = TRUE. The function block MMC_MoveVelocity reports an error indicating the busy MMC_Stop command.

Figure 5-45: Combination of two blocks for MMC_Stop with their Timing diagram – Example

5.7. Single Axis Administrative Control

Administrative function blocks are blocks where no driving motion is involved. The following single axis administrative function blocks are described:

Single Axis
MMC_AxisLink
MMC_AxisUnLink
MMC_GetFBDepth
MMC_GetTotalFbDepth
MMC_GlobalReadBoolParameter
MMC_GlobalReadParameter
MMC_GlobalWriteBoolParameter
MMC_GlobalWriteParameter
MMC_PositionProfile
MMC_Power
MMC_ReadActualPosition
MMC_ReadActualTorque
MMC_ReadActualVelocity
MMC_ReadAxisError
MMC_ReadBoolParameter
MMC_ReadDigitalInput(s)
MMC_ReadDigitalOutputs
MMC_ReadDigitalOutputs32Bit
MMC_ReadParameter
MMC_ReadStatus
MMC_Reset
MMC_SetOverride
MMC_SetPosition
MMC_TouchProbeDisable
MMC_TouchProbeEnable
MMC_WriteBoolParameter

MMC_WriteDigitalOutputs
MMC_WriteDigitalOutputs32Bit
MMC_WriteParameter

5.7.1. Elmo SuperImposed Motion

Super Imposing a motion is the ability to impose an additional profiler to the ongoing motion. This is widely used in sitautions where the end position is unknown and requires changes during the motion, without crossing the initial location. The imposed Position, and Velocity are added to the ongoing motion.

Figure 5-46 Elmo Superimposed Motion

The functions MMC_AxisLink and MMC_AxisUnlink perform this operation. This ability to perform SuperImposed motions is also integrated as part of the GMAS Script Manager functions in the EAS application. The Imposed motion is always a Virtual axis, and this can be Recorded in EAS, before and after the Superimposed is set, as shown in Figure 5-47.

Figure 5-47 Elmo Superimposed Motion Recorded in EAS

5.7.2. Special Function

The special function MMC_GETREACTORSTATISTICS communicates between the CANbus/EtherCAT and the single axis control, and is an administrative function block for the following:

- MMC_ID_AXIS_RELATE_MASK
- eMMC_FUNC_ID_AXIS_RELATED_DS301
- eMMC_FUNC_ID_AXIS_RELATED_DS401
- eMMC_FUNC_ID_AXIS_RELATED_DS402
- eMMC_FUNC_ID_AXIS_RELATED_DS406

5.7.3. MMC_GetFBDepth

Sends a command to receive the number of function blocks in the Node Queue waiting to be activated, or presently activated, which did not receive the Done or Abort status.

```
MMC_LIB_API int MMC_GetFbDepthCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GETFBDEPTH_IN* pInParam,
OUT MMC_GETFBDEPTH_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GETFBDEPTH_IN** input data structure using the MMC_GetFBDepth function.

pOutParam

Points to the **MMC_GETFBDEPTH_OUT** output structure receiving information as a result of calling the MMC_GetFBDepth function.

Remarks

None

Scope

All

MMC_GETFBDEPTH_IN Structure

```
typedef struct{
unsigned int uiHndl;
}MMC_GETFBDEPTH_IN;
```

Parameters

uiHndl

Returned function block handle. **Not in use at this moment.**

MMC_GETFBDEPTH_OUT Structure

```
typedef struct{
unsigned int  uiFbInQ;
unsigned short usStatus;
short usErrorID;
}MMC_GETFBDEPTH_OUT;
```

Parameters

uiFbInQ

Actual numbr of function blocks in queue. Any +ve integer value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-46 describes the function block for MMC_GetFBDepth.

Figure 5-48: MMC_GetFBDepth function block

5.7.3.2. Function Block Code Example

```
int rc;
MMC_GETFBDEPTH_IN      stGetFBDepth_in;
MMC_GETFBDEPTH_OUT     stGetFBDepth_out;
//
// Inserting the structure parameters:
stGetFBDepth_in.uiHndl     = 100.0;      // Returned function block handle

//
rc = MMC_GetFbDepthCmd (hConn, iAxisRef, &stGetFBDepth_in, &stGetFBDepth_out);
printf("FB Depth[%ld] ErrId[%d]\n", (long int)stGetFBDepth_out.uiFbInQ,
(short)stGetFBDepth_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.7.4. MMC_GetTotalFbDepth

Sends a command to receive the total number of function blocks in the Node Queue, waiting to be activated, presently activated, and previously activated but not released to the function block pool.

```
MMC_LIB_API int MMC_GetTotalFbDepthCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GETFBDEPTH_IN* pInParam,
OUT MMC_GETFBDEPTH_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GETFBDEPTH_IN** input data structure using the MMC_GetFBDepth function.

pOutParam

Points to the **MMC_GETFBDEPTH_OUT** output structure receiving information as a result of calling the MMC_GetFBDepth function.

Remarks

None

Scope

All

MMC_GETFBDEPTH_IN Structure

```
typedef struct{
unsigned int uiHndl;
}MMC_GETFBDEPTH_IN;
```

Parameters

uiHndl

Returned function block handle. **Not in use at this moment.**

MMC_GETFBDEPTH_OUT Structure

```
typedef struct{
unsigned int  uiFbInQ;
unsigned short usStatus;
short usErrorID;
}MMC_GETFBDEPTH_OUT;
```

Parameters

uiFbInQ

Actual number of function blocks in queue. Any +ve integer value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-46 describes the function block for MMC_GetFBDepth.

Figure 5-49: MMC_GetFBDepth function block

5.7.4.2. Function Block Code Example

```
int rc;
MMC_GETFBDEPTH_IN      stGetFBDepth_in;
MMC_GETFBDEPTH_OUT     stGetFBDepth_out;
//
// Inserting the structure parameters:
stGetFBDepth_in.uiHndl     = 100.0;      // Returned function block handle

//
rc = MMC_GetFbDepthCmd (hConn, iAxisRef, &stGetFBDepth_in, &stGetFBDepth_out);
printf("FB Depth[%ld] ErrId[%d]\n", (long int)stGetFBDepth_out.uiFbInQ,
(short)stGetFBDepth_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.7.4.3. Implementation Example

5.7.5. MMC_Power

Controls the power stage (On or Off).

```
MMC_LIB_API int MMC_PowerCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_POWER_IN* pInParam,
OUT MMC_POWER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_POWER_IN** input data structure using the MMC_Power function.

pOutParam

Points to the **MMC_POWER_OUT** output structure receiving information as a result of calling the MMC_Power function.

Remarks

It is possible to set an error variable when the Command is TRUE for a while and the Status remains false with a Timer function block and an AND Function (with inverted Status input). It indicates that there is a hardware problem with the power stage.

If power fails (also during operation) it will generate a transition to the ErrorStop state

Enable_Positive and Enable_Negative are both level triggered

Scope

MMC_Power can be set to ON or OFF. If the MC_Power function block is called when Enable = TRUE, while being in Disabled, the axis state changes to StandStill.

MMC_POWER_IN Structure

```
typedef struct{
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucEnable;
unsigned char ucEnablePositive;
unsigned char ucEnableNegative;
unsigned char ucExecute;
}MMC_POWER_IN;
```

Parameters

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucEnable

Get the value of the parameter continuously while enabled. As long as Enable is true, power is enabled. Character values of 0, 1 integers.

ucEnablePositive

As long as Enable is true, permits motion in positive direction. Boolean values TRUE/FALSE accepted.

ucEnableNegative

As long as Enable is true, permits motion in negative direction (_Positive & _Negative can both be true). Boolean values TRUE/FALSE accepted.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_POWER_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_POWER_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-48 describes the function block for MMC_Power

Figure 5-50: MMC_Power function block

5.7.5.2. Function Block Code Example

```
int rc;
MMC_POWER_IN       stPower_in;
MMC_POWER_OUT      stPower_out;
//
// Inserting the structure parameters:
stPower_in.ucEnable             = 1;                  //Enabled
stPower_in.ucEnablePositive     = 1;                  //Permits motion in positive direction
stPower_in.ucEnableNegative     = 0;                  //Does not permit motion in negative
direction
stPower_in.eBufferMode          = MC_BUFFERED_MODE;   //MC_BUFFERED_MODE_ENUM Defines the
behavior of the axis
stPower_in.ucExecute            = 1;
//
rc = MMC_PowerCmd (hConn, iAxisRef, &stPower_in, &stPower_out);
if (rc != 0)
{
    HandleError();
}
```

5.7.5.3. Implementation Example

5.7.6. MMC_PositionProfile

Describes the position profile of an axis.

```
MMC_LIB_API int MMC_PositionProfileCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_POSITIONPROFILE_IN* pInParam,
OUT MMC_POSITIONPROFILE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_POSITIONPROFILE_IN** input data structure using the MMC_PositionProfile function.

pOutParam

Points to the **MMC_POSITIONPROFILE_OUT** output structure receiving information as a result of calling the MMC_PositionProfile function.

Remarks

None

Scope

All

MMC_POSITIONPROFILE_IN Structure

```
typedef struct mmc_positionprofile_in{
MC_BUFFERED_MODE_ENUM eBufferMode;
MC_PATH_REF hMemHandle;
unsigned char ucExecute;
}MMC_POSITIONPROFILE_IN;
```

Parameters

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

hMemHandle

MC_PATH_REF enumerator handle to a journal entry where the pointer to the shared memory is located. MC_PATH_REF is the journal entry path reference.

hMemHandle can have integer values.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_POSITIONPROFILE_OUT Structure

```
typedef struct mmc_positionprofile_out{
unsigned short usStatus;
unsigned short usErrorID;
}MMC_POSITIONPROFILE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-49 describes the function block for MMC_PositionProfile.

Figure 5-51: MMC_PositionProfile function block

5.7.6.2. Function Block Code Example

5.7.6.3. Implementation Example

5.7.7. MMC_ReadActualPosition

Returns the actual position of the controlled axis.

```
MMC_LIB_API int MMC_ReadActualPositionCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READACTUALPOSITION_IN* pInParam,
OUT MMC_READACTUALPOSITION_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READACTUALPOSITION_IN** input data structure using the MMC_ReadActualPosition function.

pOutParam

Points to the **MMC_READACTUALPOSITION_OUT** output structure receiving information as a result of calling the MMC_ReadActualPosition function.

Remarks

For CAN only, depending on the mode, the position may not be updated. If Distributed motion mode is in operation, then this will depend whether the parameter was mapped via PDO mapping. Refer to the PDO3, 4 configuration mapping functions in section **12.7**.

For EtherCAT, the actual position must be part of the Input Group configured.

Scope

All

MMC_READACTUALPOSITION_IN Structure

```
typedef struct{
unsigned char ucEnable;
}MMC_READACTUALPOSITION_IN;
```

Parameters

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READACTUALPOSITION_OUT Structure

```
typedef struct{
double dbPosition;
unsigned short usStatus;
short usErrorID;
}MMC_READACTUALPOSITION_OUT;
```

Parameters

dbPosition

New absolute position. Any -ve or +ve double values in axis's unit [u]

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-50 describes the function block for MMC_ReadActualPosition

Figure 5-52: MMC_ReadActualPosition function block

5.7.7.2. Function Block Code Example

```
int rc ;
MMC_READACTUALPOSITION_IN       stReadActPos_in ;
MMC_READACTUALPOSITION_OUT      stReadActPos_out ;
//
// Inserting the structure parameters:
stReadActPos_in.ucEnable = 1; //Enabled
//
rc = MMC_ReadActualPositionCmd (hConn, iAxisRef,&stReadActPos_in, &stReadActPos_out);
printf("Actual Position[%ld] ErrId[%d]\n", (long int)stReadActPos_out.dbPosition,
(short)stReadActPos_out.usErrorID);
if (rc != 0)
{
    HandleError() ;
}
```

5.7.8. MMC_ReadActualTorque

Returns the actual torque value or force for a controlled axis, as long as Enable is set.

```
MMC_LIB_API int MMC_ReadActualTorqueCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READACTUALTORQUE_IN* pInParam,
OUT MMC_READACTUALTORQUE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READACTUALTORQUE_IN** input data structure using the MMC_ReadActualTorque function.

pOutParam

Points to the **MMC_READACTUALTORQUE_OUT** output structure receiving information as a result of calling the MMC_ReadActualTorque function.

Remarks

For CAN only, depending on the mode, the position may not be updated. The torque is mapped and not the position when in all opModes, not just in interpolated mode.

If Distributed motion mode is in operation, then this will depend whether the parameter was mapped via PDO mapping. Refer to the PDO3, 4 configuration mapping functions in section **11.4.3**.

For EtherCAT, the actual position must be part of the Input Group configured.

Scope

The usStatus mode Done is TRUE when the data-output Torque is valid. If Enable is Reset, the data loses its validity, and Done is also reset, no matter if new data is available.

The output dActualTorque can be a signed value.

MMC_READACTUALTORQUE_IN Structure

```
typedef struct{
unsigned char ucEnable;
}MMC_READACTUALTORQUE_IN;
```

Parameters

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READACTUALTORQUE_OUT Structure

```
typedef struct{
double dActualTorque;
unsigned short usStatus;
short usErrorID;
unsigned char ucValid;
}MMC_READACTUALTORQUE_OUT;
```

Parameters

dActualTorque

The value of the actual torque or force. Any -ve or +ve double values in torque units.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Displays an error code as -ve or +ve integers.

ucValid

Values inputted are valid or not. Boolean values TRUE/FALSE accepted.

Figure 5-51 describes the function block for MMC_ReadActualTorque

Figure 5-53: MMC_ReadActualTorque function block

5.7.8.2. Function Block Code Example

```
int rc;
MMC_READACTUALTORQUE_IN         stReadActTorque_in;
MMC_READACTUALTORQUE_OUT        stReadActTorque_out;
//
// Inserting the structure parameters:
stReadActTorque_in.ucEnable  = 1;        // Enabled
//
rc = MMC_ReadActualTorqueCmd (hConn, iAxisRef, &stReadActTorque_in, &stReadActTorque_out);
printf("Actual Torque[%ld] ErrId[%d]\n", (long int)stReadActTorque_out.dActualTorque,
(short)stReadActTorque_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.7.9. MMC_ReadActualVelocity

Returns the actual velocity value, as long as Enable is set.

```
MMC_LIB_API int MMC_ReadActualPositionCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READACTUALPOSITION_IN* pInParam,
OUT MMC_READACTUALPOSITION_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READACTUALVELOCITY_IN** input data structure using the MMC_ReadActualVelocity function.

pOutParam

Points to the **MMC_READACTUALVELOCITY_OUT** output structure receiving information as a result of calling the MMC_ReadActualVelocity function.

Remarks

For CAN only, depending on the mode, the position may not be updated. If Distributed motion mode is in operation, then this will depend whether the parameter was mapped via PDO mapping. Refer to the PDO3, 4 configuration mapping functions in section **12.7**.

For EtherCAT, the actual position must be part of the Input Group configured.

Scope

Valid is true when the data-output Velocity is valid. If Enable is Reset, the data loses its validity, and all outputs are reset, no matter if new data is available.

The output dVelocity can be a signed value.

MMC_READACTUALVELOCITY_IN Structure

```
typedef struct{
unsigned char ucEnable;
}MMC_READACTUALVELOCITY_IN;
```

Parameters

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READACTUALVELOCITY_OUT Structure

```
typedef struct{
double dVelocity;
unsigned short usStatus;
short usErrorID;
}MMC_READACTUALVELOCITY_OUT;
```

Parameters

dVelocity

The value of the actual velocity. Any positive double value in u/s units

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-52 describes the function block for MMC_ReadActualVelocity

Figure 5-54: MMC_ReadActualVelocity function block

5.7.9.2. Function Block Code Example

```
int rc;
MMC_READACTUALVELOCITY_IN       stReadActVel_in;
MMC_READACTUALVELOCITY_OUT      stReadActVel_out;
//
// Inserting the structure parameters:
stReadActVel_in.ucEnable = 1;
//
rc = MMC_ReadActualVelocityCmd (hConn, iAxisRef, &stReadActVel_in, &stReadActVel_out);
printf("Actual Velocity[%ld] ErrId[%d]\n", (long int)stReadActVel_out.dVelocity,
(short)stReadActVel_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.7.10. MMC_ReadAxisError

Displays general axis errors not relating to the function blocks e.g. axis errors, drive errors, communication errors.

```
MMC_LIB_API int MMC_ReadAxisError(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READAXISERROR_IN* pInParam,
OUT MMC_READAXISERROR_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	
	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READAXISERROR_IN** input data structure using the MMC_ReadAxisError function.

pOutParam

Points to the **MMC_READAXISERROR_OUT** output structure receiving information as a result of calling the MMC_ReadAxisError function.

Remarks

None

Scope

All

MMC_READAXISERROR_IN Structure

```
typedef struct{
unsigned char ucEnable;
}MMC_READAXISERROR_IN;
```

Parameters

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READAXISERROR_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned short usAxisErrorID;
unsigned short usLastEmergencyErrCode;
}MMC_READAXISERROR_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

usAxisErrorID

Returns the axis error bitwise ID defined by the following enumerators. Bitwise ID error code:

ID	Enumerator
0x1	MMC_ERR_TYPE_FAULT_BIT
0x2	MMC_ERR_TYPE_HEARTBEAT
0x4	MMC_ERR_TYPE_EMERGENCY
0x8	MMC_ERR_TYPE_COMM
0x10	MMC_ERR_TYPE_CFG_FILE

usLastEmergencyErrCode

Last emergency DS-402 error code that occurred in the system. Refer to the DS-402 Elmo emergency codes in the following:

- CANopen DS-301 Implementation Guide Chapter 6
- CANopen DSP-402 Implementation Guide
- SimplIQ Command Reference Guide or Gold Command Reference Guide

Figure 5-53 describes the function block for MMC_ReadAxisError

Figure 5-55: MMC_ReadAxisError function block

5.7.10.2. Function Block Code Example

```
int rc ;
MMC_READAXISERROR_IN   stReadAxisError_in;
MMC_READAXISERROR_OUT  stReadAxisError_out;
//
// Inserting the structure parameters:
stReadAxisError_in.ucEnable  = 1;        // Enabled
//
rc = MMC_ReadAxisError (hConn, iAxisRef, &stReadAxisError_in, &stReadAxisError_out) ;
printf("Axis Error[%ld] ErrId[%d]\n", (long int)stReadAxisError_out.usAxisErrorID,
(short)stReadAxisError_out.usErrorID, (short)stReadAxisError_out.usLastEmergencyErrCode);
if (rc != 0)
{
    HandleError() ;
}
```

5.7.11. MMC_ReadBoolParameter

Returns the value of a vendor specific with datatype unsigned long or un signed int.

```
MMC_LIB_API int MMC_ReadBoolParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READBOOLPARAMETER_IN* pInParam,
OUT MMC_READBOOLPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READBOOLPARAMETER_IN** input data structure using the MMC_ReadBoolParameter function.

pOutParam

Points to the **MMC_READBOOLPARAMETER_OUT** output structure receiving information as a result of calling the MMC_ReadBoolParameter function.

Remarks

None

Scope

All

MMC_READBOOLPARAMETER_IN Structure

```
typedef struct{
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_READBOOLPARAMETER_IN;
```

Parameters

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READBOOLPARAMETER_OUT Structure

```
typedef struct{
long lValue;
unsigned short usStatus;
short usErrorID;
}MMC_READBOOLPARAMETER_OUT;
```

Parameters

lValue

Boolean parameters integer value

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-54 describes the function block for MMC_ReadBoolParameter

Figure 5-56: MMC_ReadBoolParameter function block

5.7.11.2. Function Block Code Example

```
int rc ;
MMC_READBOOLPARAMETER_IN   stReadBoolParam_in;
MMC_READBOOLPARAMETER_OUT  stReadBoolParam_out;
//
// Inserting the structure parameters:
stReadBoolParam_in.eParameterNumber      = MMC_AXIS_MODE_PARAM; //MMC_PARAMETER_LIST_ENUM
enumerator
stReadBoolParam_in.iParameterArrIndex    = 0;                   //Array index parameter
stReadBoolParam_in.ucEnable              = 1;                   //Enabled
//
rc = MMC_ReadBoolParameter (hConn, iAxisRef, &stReadBoolParam_in, &stReadBoolParam_out) ;
printf("Boolean Parameter[%ld] ErrId[%d]\n", (long int)stReadBoolParam_out.lValue,
(short)stReadBoolParam_out.usErrorID);
if (rc != 0)
{
    HandleError() ;
}
```

5.7.12. MMC_GlobalReadBoolParameter

Returns the value of a vendor global parameter with datatype unsigned long or un signed int.

```
MMC_LIB_API int MMC_GlobalReadBoolParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_READBOOLPARAMETER_IN* pInParam,
OUT MMC_READBOOLPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	
	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READBOOLPARAMETER_IN** input data structure using the MMC_GlobalReadBoolParameter function.

pOutParam

Points to the **MMC_READBOOLPARAMETER_OUT** output structure receiving information as a result of calling the MMC_GlobalReadBoolParameter function.

Remarks

None

Scope

All

MMC_READBOOLPARAMETER_IN Structure

```
typedef struct{
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_READBOOLPARAMETER_IN;
```

Parameters

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READBOOLPARAMETER_OUT Structure

```
typedef struct{
long lValue;
unsigned short usStatus;
short usErrorID;
}MMC_READBOOLPARAMETER_OUT;
```

Parameters

lValue

Boolean parameters integer value

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-55 describes the function block for MMC_GlobalReadBoolParameter

Figure 5-57: MMC_GlobalReadBoolParameter function block

5.7.12.2. Function Block Code Example

```
int rc ;
MMC_READBOOLPARAMETER_IN   stReadBoolParam_in;
MMC_READBOOLPARAMETER_OUT  stReadBoolParam_out;
//
// Inserting the structure parameters:
stReadBoolParam_in.eParameterNumber      = MMC_CYCLE_TIME_PARAM;//MMC_PARAMETER_LIST_ENUM
enumerator
stReadBoolParam_in.iParameterArrIndex    = 0;                   //Array index parameter
stReadBoolParam_in.ucEnable              = 1;                   //Enabled
//
rc = MMC_ReadBoolParameter (hConn, iAxisRef, &stReadBoolParam_in, &stReadBoolParam_out) ;
printf("Global Boolean Parameter[%ld] ErrId[%d]\n", (long int)stReadBoolParam_out.lValue,
(short)stReadBoolParam_out.usErrorID);
if (rc != 0)
{
    HandleError() ;
}
```

5.7.13. MMC_ReadDigitalInput(s)

Accesses the value of the input, referenced by the datatype MMC_Axis_Ref. It provides the Boolean value of the referenced input.

```
MMC_LIB_API int MMC_ReadDigitalInputCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READDIGITALINPUT_IN* pInParam,
OUT MMC_READDIGITALINPUT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READDIGITALINPUT_IN** input data structure using the MMC_ReadDigitalInput function.

pOutParam

Points to the **MMC_READDIGITALINPUT_OUT** output structure receiving information as a result of calling the MMC_ReadDigitalInput function.

Remarks

For CAN, depending on the mode, the Input(s) may not be updated.

For EtherCAT, the actual Input(s) must be part of the Input Group configured.

This will depend whether the parameter was mapped via PDO mapping. Refer to the PDO3, 4 configuration mapping functions in section **12.7**.

Scope

All

MMC_READDIGITALINPUT_IN Structure

```
typedef struct{
int iInputNumber;
unsigned char ucEnable;
}MMC_READDIGITALINPUT_IN;
```

Parameters

iInputNumber

Selects the input depending on the drive. Can be part of MMC_Axis_Ref, if only one single input is referenced. +ve integer value

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READDIGITALINPUT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned char ucValue;
}MMC_READDIGITALINPUT_OUT;
```

Parameters

ucValue

Selected value of the input signal. +ve integer value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

MMC_ReadDigitalInputs

```
MMC_LIB_API int MMC_ReadDigitalInputsCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READDIGITALINPUTS_IN* pInParam,
OUT MMC_READDIGITALINPUTS_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READDIGITALINPUTS_IN** input data structure using the MMC_ReadDigitalInputs function.

pOutParam

Points to the **MMC_READDIGITALINPUTS_OUT** output structure receiving information as a result of calling the MMC_ReadDigitalInputs function.

MMC_READDIGITALINPUTS_IN Structure

```
typedef struct{
unsigned char ucEnable;
}MMC_READDIGITALINPUTS_IN;
```

Parameters

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READDIGITALINPUTS_OUT Structure

```
typedef struct{
unsigned long ulValue;
unsigned short usStatus;
short usErrorID;
}MMC_READDIGITALINPUTS_OUT;
```

Parameters

ulValue

Total value of all the inputs together. +ve 32 bit bitwise numeric value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-56 describes the function block for MMC_ReadDigitalInput

Figure 5-58: MMC_ReadDigitalInput function block

Figure 5-57 describes the function block for MMC_ReadDigitalInputs

Figure 5-59: MMC_ReadDigitalInputs function block

5.7.13.2. Function Block Code Example

```
int rc;
MMC_READDIGITALINPUT_IN         stReadDigInput_in;
MMC_READDIGITALINPUT_OUT        stReadDigInput_out;
//
// Inserting the structure parameters:
stReadDigInput_in.iInputNumber  = 16;    //Selects the input depending on the drive
stReadDigInput_in.ucEnable      = 1;     //Enabled
//
rc = MMC_ReadDigitalInputCmd (hConn, iAxisRef, &stReadDigInput_in, &stReadDigInput_out);
printf("Digital Input[%ld] ErrId[%d]\n", (long int)stReadDigInput_out.ucValue,
(short)stReadDigInput_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.7.14. MMC_ReadDigitalOutputs

Reads the actual digital outputs for the specific node.

```
MMC_LIB_API int MMC_ReadDigitalOutputs(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READDIGITALOUTPUT_IN* pInParam,
OUT MMC_READDIGITALOUTPUT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READDIGITALOUTPUT_IN** input data structure using the MMC_ReadDigitalOutputs function.

pOutParam

Points to the **MMC_READDIGITALOUTPUT_OUT** output structure receiving information, as a result of calling the MMC_ReadDigitalOutputs function.

Remarks

Provides the value of the digital outputs, referenced by the datatype MC_OutputRef. It provides the Boolean value of the referenced outputs.

It is not guaranteed that the digital signal will be seen by the function block, as a short pulse on the digital output could be completed before the next function block cycle occurs.

Scope

All

MMC_READDIGITALOUTPUT_IN Structure

```
typedef struct MMC_READDIGITALOUTPUT_IN{
int iOutputNumber;
unsigned char ucEnable;
}MMC_READDIGITALOUTPUT_IN;
```

Parameters

iOutputNumber

Selects the output. Can be part of MC_OutputRef, if only one single output is referenced. +ve integer value.

Note: **This parameter is not in use at this time.**

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READDIGITALOUTPUT_OUT Structure

```
typedef struct MMC_READDIGITALOUTPUT_OUT{
unsigned short usStatus;
short usErrorID;
unsigned char ucValue;
}MMC_READDIGITALOUTPUT_OUT;
```

Parameters

ucValue

Selected value of the input signal. +ve integer value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-58 describes the function block for MMC_ReadDigitalOutput.

Figure 5-60: MMC_ReadDigitalOutput function block

5.7.14.2. Function Block Code Example

```
int rc;
MMC_READDIGITALOUTPUT_IN   stReadDigOutput_in;
MMC_READDIGITALOUTPUT_OUT  stReadDigOutput_out;
//
// Inserting the structure parameters:
stReadDigOutput_in.iOutputNumber    = 16;    //Selects the output depending on the drive
stReadDigOutput_in.ucEnable         = 1;     //Enabled
//
rc = MMC_ReadDigitalOutputs (hConn, iAxisRef, &stReadDigOutput_in, &stReadDigOutput_out);
printf("Digital Outputs[%ld] ErrId[%d]\n", (long int)stReadDigOutput_out.ucValue,
(short)stReadDigOutput_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.7.15. MMC_ReadDigitalOutputs32Bit

Reads the actual 32 bit digital output for the specific node

```
MMC_LIB_API int MMC_ReadDigitalOutputs32Bit(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READDIGITALOUTPUT32Bit_IN*pInParam,
OUT MMC_READDIGITALOUTPUT32Bit_OUT*pOutParam
);
```

Motion Mode NC - Supported Distributed - Supported

Source GMAS\includes\MMC_PLCopen_single_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READDIGITALOUTPUT32Bit_IN** input data structure using the MMC_ReadDigitalOutputs32Bit function.

pOutParam

Points to the **MMC_READDIGITALOUTPUT32Bit_OUT** output structure receiving information, as a result of calling the MMC_ReadDigitalOutputs32Bit function.

Remarks

Provides the value of the 32 bit digital outputs, referenced by the datatype MC_OutputRef. It provides the Boolean value of the referenced outputs.

The relevant function must be supported for digital outputs.

Scope

All

MMC_READDIGITALOUTPUT32Bit_IN Structure

```
typedef struct MMC_READDIGITALOUTPUT32Bit_IN{
int iOutputNumber;
unsigned char ucEnable;
}MMC_READDIGITALOUTPUT32Bit_IN;
```

Parameters

iOutputNumber

Selects the output. Can be part of MC_OutputRef, if only one single output is referenced. +ve integer value.

Note: **This parameter is not in use at this time.**

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READDIGITALOUTPUT32Bit_OUT Structure

```
typedef struct MMC_READDIGITALOUTPUT32Bit_OUT{
unsigned short usStatus;
unsigned short usErrorID;
unsigned long ulValue;
}MMC_READDIGITALOUTPUT32Bit_OUT;
```

Parameters

ulValue

Total value of all the inputs together. +ve 32 bit bitwise numeric value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-59 describes the function block for MMC_ReadDigitalOutputs32Bit

Figure 5-61: MMC_ReadDigitalOutputs32Bit function block

5.7.15.2. Function Block Code Example

```
int rc;
MMC_READDIGITALOUTPUT32Bit_IN       stReadDigOutput32Bit_in;
MMC_READDIGITALOUTPUT32Bit_OUT      stReadDigOutput32Bit_out;
//
// Inserting the structure parameters:
stReadDigOutput32Bit_in.iOutputNumber    = 16;    //Selects the output depending on the drive
stReadDigOutput32Bit_in.ucEnable         = 1;     //Enabled
//
rc = MMC_ReadDigitalOutputs32Bit (hConn, iAxisRef, &stReadDigOutput32Bit_in,
&stReadDigOutput32Bit_out);
printf("Digital Outputs 32Bit[%ld] ErrId[%d]\n", (long int)stReadDigOutput32Bit_out.ulValue,
(short)stReadDigOutput32Bit_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.7.16. MMC_ReadParameter

Returns the value of a vendor specific parameter.

```
MMC_LIB_API int MMC_ReadParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READPARAMETER_IN* pInParam,
OUT MMC_READPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READPARAMETER_IN** input data structure using the MMC_ReadParameter function.

pOutParam

Points to the **MMC_READPARAMETER_OUT** output structure receiving information as a result of calling the MMC_ReadParameter function.

Remarks

None

Scope

All

MMC_READPARAMETER_IN Structure

```
typedef struct{
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_READPARAMETER_IN;
```

Parameters

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READPARAMETER_OUT Structure

```
typedef struct{
double dbValue;
unsigned short usStatus;
short usErrorID;
}MMC_READPARAMETER_OUT;
```

Parameters

dbValue

Output of the specific parameter. Any Double value.

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-60 describes the function block for MMC_ReadParameter

Figure 5-62: MMC_ReadParameter function block

5.7.16.2. Function Block Code Example

```
int rc;
MMC_READPARAMETER_IN   stReadParam_in;
MMC_READPARAMETER_OUT  stReadParam_out;
//
// Inserting the structure parameters:
stReadParam_in.ucEnable              = 1;                    //Enabled
stReadParam_in.iParameterArrIndex    = 0;                    //Parameter array index
stReadParam_in.eParameterNumber      = MMC_SET_POSITION_PARAM;  //Axis parameter value
//
rc = MMC_ReadParameter (hConn, iAxisRef, &stReadParam_in, &stReadParam_out);
printf("Read Parameter[%ld] ErrId[%d]\n", (long int)stReadParam_out.dbValue,
(short)stReadParam_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.7.17. MMC_GlobalReadParameter

Returns the value of a vendor global parameter.

```
MMC_LIB_API int MMC_GlobalReadParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_READPARAMETER_IN* pInParam,
OUT MMC_READPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	
	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_READPARAMETER_IN** input data structure using the MMC_GlobalReadParameter function.

pOutParam

Points to the **MMC_READPARAMETER_OUT** output structure receiving information as a result of calling the MMC_GlobalReadParameter function.

Remarks

None

Scope

All

MMC_READPARAMETER_IN Structure

```
typedef struct{
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_READPARAMETER_IN;
```

Parameters

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READPARAMETER_OUT Structure

```
typedef struct{
double dbValue;
unsigned short usStatus;
short usErrorID;
}MMC_READPARAMETER_OUT;
```

Parameters

dbValue

Output of specific parameter. Any Double value.

usStatus

Bitwise returned command status with the following values:

Aborted, Done, CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-61 describes the function block for MMC_GlobalReadParameter

Figure 5-63: MMC_GlobalReadParameter function block

5.7.17.2. Function Block Code Example

```
int rc;
MMC_READPARAMETER_IN   stReadParam_in;
MMC_READPARAMETER_OUT  stReadParam_out;
//
// Inserting the structure parameters:
stReadParam_in.ucEnable              = 1;                    //Enabled
stReadParam_in.iParameterArrIndex    = 0;                    //Parameter array index
stReadParam_in.eParameterNumber      = MMC_CYCLE_TIME_PARAM;//Axis parameter value
//
rc = MMC_ReadParameter (hConn, iAxisRef, &stReadParam_in, &stReadParam_out);
printf("Global Read Parameter[%ld] ErrId[%d]\n", (long int)stReadParam_out.dbValue,
(short)stReadParam_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.7.18. MMC_ReadStatus

Returns details of the state diagram status for the selected axis.

```
MMC_LIB_API int MMC_ReadStatusCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READSTATUS_IN* pInParam,
OUT MMC_READSTATUS_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READSTATUS_IN** input data structure using the MMC_ReadStatus function.

pOutParam

Points to the **MMC_READSTATUS_OUT** output structure receiving information as a result of calling the MMC_ReadStatus function.

Remarks

None

Scope

All

MMC_READSTATUS_IN Structure

```
typedef struct{
unsigned int  uiHndlr;
unsigned char ucEnable;
}MMC_READSTATUS_IN;
```

Parameters

uiHndlr

Requested axis handle integer. Any +ve integer number

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READSTATUS_OUT Structure

```
typedef struct{
unsigned long ulState;
unsigned short usStatus;
short usErrorID;
unsigned short usAxisErrorID;
unsigned short usStatusWord;
}MMC_READSTATUS_OUT;
```

Parameters

ulState

Returned command status. Refer to the sub-section **5.4 Axis Status** for the appropriate parameter value received and its explanation. Bitwise value returned.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

usAxisErrorID

Returns the axis error bitwise ID defined by the following enumerators. Bitwise ID error code:

ID	Enumerator
0x1	MMC_ERR_TYPE_FAULT_BIT
0x2	MMC_ERR_TYPE_HEARTBEAT
0x4	MMC_ERR_TYPE_EMERGENCY
0x8	MMC_ERR_TYPE_COMM
0x10	MMC_ERR_TYPE_CFG_FILE

usStatusWord

Drive Status text. Any text characters.

Figure 5-62 describes the function block for MMC_ReadStatus

Figure 5-64: MMC_ReadStatus function block

5.7.18.2. Function Block Code Example

```
int rc ;
MMC_READSTATUS_IN      stReadStatus_in;
MMC_READSTATUS_OUT     stReadStatus_out;
//
// Inserting the structure parameters:
stReadStatus_in.uiHndlr     = 251;    //Requested axis handle integer
stReadStatus_in.ucEnable    = 1;      //Enabled
//
rc = MMC_ReadStatusCmd (hConn, iAxisRef, &stReadStatus_in, &stReadStatus_out);
printf("Read Status[%ld] ErrId[%d]\n", stReadStatus_out.ulState,
(short)stReadStatus_out.usErrorID);

if (rc != 0)
{
    HandleError() ;
}
```

5.7.19. MMC_Reset

Produces the transition from the state ErrorStop to StandStill or Disabled by resetting all internal axis-related errors.

```
MMC_LIB_API int MMC_Reset(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_RESET_IN* pInParam,
OUT MMC_RESET_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_RESET_IN** input data structure using the MMC_Reset function.

pOutParam

Points to the **MMC_RESET_OUT** output structure receiving information as a result of calling the MMC_Reset function.

Remarks

However, it does not affect the output of the function block instances.

This function must be called in order to exit the ErrorStop state.

Scope

All

MMC_RESET_IN Structure

```
typedef struct{
unsigned char ucExecute;
}MMC_RESET_IN;
```

Parameters

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_RESET_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_RESET_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-63 describes the function block for MMC_Reset

Figure 5-65: MMC_Reset function block

5.7.19.2. Function Block Code Example

```
int rc;
MMC_RESET_IN      stReset_in;
MMC_RESET_OUT     stReset_out;
//
// Inserting the structure parameters:
stReset_in.ucExecute   = 1;     // Start the execution command

//
rc = MMC_Reset (hConn, iAxisRef, &stReset_in, &stReset_out);
if (rc != 0)
{
    HandleError();
}
```

5.7.20. MMC_SetOverride

Sets the values of override for the whole axis, including all functions that are operating on that axis.

```
MMC_LIB_API int MMC_SetOverrideCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_SETOVERRIDE_IN* pInParam,
OUT MMC_SETOVERRIDE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	**Function** GMAS\includes\MMC_PLCopen_single_API.h **Structures** GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_SETOVERRIDE_IN** input data structure using the MMC_SetOverride function.

pOutParam

Points to the **MMC_SETOVERRIDE_OUT** output structure receiving information as a result of calling the MMC_SetOverride function.

Remarks

The override parameters are applied to the commanded velocity, acceleration, deceleration, and jerk of the relevant Motion function block.

MMC_SetOverride does not influence the state diagram of the axis. The following should be noted:

- The Input *fAccFactor* acts on positive and negative acceleration (deceleration).
- This function block sets the override factor, which is valid until a new override is set.
- The default values of the override factor are 1.0. The value of the overrides can be between 0.0 and 1.0. The behavior of values > 1.0 is vendor specific. Values < 0.0 are not allowed. The value 0.0 is not allowed for *fAccFactor* and *fJerkFactor*.
- The value 0.0 set to the *fVelFactor* stops the axis without bringing it to the state Standstill.

- Override does not act on slave axes. (Axes in the state synchronized motion).
- *fVelFactor* can be changed at any time and acts directly on the ongoing motion.
- If in Discrete motion, reducing the fAccFactor and/or fJerkFactor can lead to a position overshoot – a possible cause of damage.

Activating this function block on an axis that is under control of MC_PositionProfile, MC_VelocityProfile, or MC_AccelerationProfile, is vendor specific.

Scope

All

MMC_SETOVERRIDE_IN Structure

```
typedef struct{
float fVelFactor;
float fAccFactor;
float fJerkFactor;
unsigned short usUpdateVelFactorIdx;
unsigned char ucEnable;
}MMC_SETOVERRIDE_IN;
```

Parameters

fVelFactor

New override factor for the velocity. Any +ve float value between [0 – 1].

fAccFactor

New override factor for the acceleration/deceleration. Any +ve float value between [0 – 1].

fJerkFactor

New override factor for the jerk. Any +ve float value between [0 – 1].

usUpdateVelFactorIdx

Index of changed velocity factor. Vendor defined. The default is 0. Has integer values of 0 - 2

This variable is not in use at this moment.

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_SETOVERRIDE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
} MMC_SETOVERRIDE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block.

Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Displays an error code as -ve or +ve integers.

Figure 5-64 describes the function block for MMC_SetOverride

Figure 5-66: MMC_SetOverride function block

5.7.20.2. Function Block Code Example

```
int rc;
MMC_SETOVERRIDE_IN          stSetOverride_in;
MMC_SETOVERRIDE_OUT         stSetOverride_out;
//
// Inserting the structure parameters:
stSetOverride_in.fAccFactor              = 0.5; // New override factor for the
acceleration/deceleration
stSetOverride_in.fJerkFactor             = 0.4; // New override factor for the jerk
stSetOverride_in.usUpdateVelFactorIdx    = 0;   // Index of changed velocity factor
stSetOverride_in.fVelFactor              = 0.8; // New override factor for the velocity
stSetOverride_in.ucEnable                = 1;   // Enabled
//
rc = MMC_SetOverrideCmd (hConn, iAxisRef, &stSetOverride_in, &stSetOverride_out);
if (rc != 0)
{
    HandleError();
}
```

5.7.21. MMC_SetPosition

Sends the Set Position command to the G-MAS for ac specific axis.

```
MMC_LIB_API int MMC_SetPositionCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_SETPOSITION_IN* pInParam,
OUT MMC_SETPOSITION_OUT* pOutParam
);
```

Motion Mode NC – Not Supported Distributed - Supported

Source GMAS\includes\MMC_PLCopen_single_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_SETPOSITION_IN** input data structure using the MMC_SetPosition function.

pOutParam

Points to the **MMC_SETPOSITION_OUT** output structure receiving information, as a result of calling the MMC_SetPosition function.

Remarks

None

Scope

All

MMC_SETPOSITION_IN Structure

```
typedef struct{
double dbPosition;
double dbModulus;
unsigned char ucPosMode;
unsigned char ucExecute;
}MMC_SETPOSITION_IN;
```

Parameters

dbPosition

Target position for the motion of the axis when conditions are met. Any -ve or +ve double values in technical unit [u].

dbModulus

The relative modulus of the axis. Any -ve or +ve double values in technical unit [u].

ucPosMode

Boolean values for the position mode can be absolute mode = 0, or relative mode = 1

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_SETPOSITION_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SETPOSITION_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-65 describes the function block for MMC_SetPosition

Figure 5-67: MMC_SetPosition function block

5.7.21.2. Function Block Code Example

5.7.22. MMC_TouchProbeEnable

Enables the touch probe to record an axis position at a trigger event.

```
MMC_LIB_API int MMC_TouchProbeEnableCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_TOUCHPROBEENABLE_IN*pInParam,
OUT MMC_TOUCHPROBEENABLE_OUT*pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_TOUCHPROBEENABLE_IN** input data structure using the MMC_TouchProbeEnable function.

pOutParam

Points to the **MMC_TOUCHPROBEENABLE_OUT** output structure receiving information as a result of calling the MMC_TouchProbeEnable function.

Remarks

This function block is intended for single shot operation, that is, the first event after the rising edge at Execute is valid for recording only. Possible following events are ignored

One function block instance should represent exactly one probing command

In case of multiple instances on the same probe and axis, the elements of TRIGGER_REF should be extended with TouchProbeID - Identification of a unique probing command – this can be linked to MC_AbortTrigger.

Scope

Supported for Gold Drives only.

User notification when touch probe is over via Callback and event.

MMC_TOUCHPROBEENABLE_IN Structure

```
typedef struct mmc_touchprobeenable_in{
unsigned char ucExecute;
unsigned char ucTriggerType;
}MMC_TOUCHPROBEENABLE_IN;
```

Parameters

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

ucTriggerType

Trigger enables the touch proble. Reference to the trigger signal source, where the trigger input may be specified by the AXIS_REF. Has the following enumerator values:

eMMC_TOUCHPROBE_POS_EDGE = 0,

eMMC_TOUCHPROBE_NEG_EDGE

MMC_TOUCHPROBEENABLE_OUT Structure

```
typedef struct mmc_touchprobeenable_out{
unsigned short usStatus;
short usErrorID;
}MMC_TOUCHPROBEENABLE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-66 describes the function block for MMC_TouchProbeEnable

Figure 5-68: MMC_TouchProbeEnable function block

5.7.22.2. Function Block Code Example

5.7.23. MMC_TouchProbeDisable

Disables the touch probe to record an axis position at a trigger event.

```
MMC_LIB_API int MMC_TouchProbeDisableCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_TOUCHPROBEDISABLE_IN* pInParam,
OUT MMC_TOUCHPROBEDISABLE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_TOUCHPROBEDISABLE_IN** input data structure using the MMC_TouchProbeDisable function.

pOutParam

Points to the **MMC_TOUCHPROBEDISABLE_OUT** output structure receiving information as a result of calling the MMC_TouchProbeDisable function.

Remarks

This function block is intended for single shot operation, that is, the first event after the rising edge at Execute is valid for recording only. Possible following events are ignored.

One function block instance should represent exactly one probing command.

In case of multiple instances on the same probe and axis, the elements of TRIGGER_REF should be extended with TouchProbeID - Identification of a unique probing command – this can be linked to MC_AbortTrigger.

Scope

Supported for Gold Drives only.

User notification when touch probe is over via Callback and event.

MMC_TOUCHPROBEDISABLE_IN Structure

```
typedef struct mmc_touchprobedisable_in{
unsigned char ucExecute;
}MMC_TOUCHPROBEDISABLE_IN;
```

Parameters

ucExecute

Start the execution command. Boolean TRUE/FALSE values.

MMC_TOUCHPROBEDISABLE_OUT Structure

```
typedef struct mmc_touchprobedisable_out{
unsigned short usStatus;
short usErrorID;
}MMC_TOUCHPROBEDISABLE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-67 describes the function block for MMC_TouchProbeDisable

Figure 5-69: MMC_TouchProbeDisable function block

5.7.23.2. Function Block Code Example

5.7.24. MMC_WriteBoolParameter

Modifies the value of a vendor specific parameter of type BOOL.

```
MMC_LIB_API int MMC_WriteBoolParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_WRITEBOOLPARAMETER_IN* pInParam,
OUT MMC_WRITEBOOLPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEBOOLPARAMETER_IN** input data structure using the MMC_WriteBoolParameter function.

pOutParam

Points to the **MMC_WRITEBOOLPARAMETER_OUT** output structure receiving information, as a result of calling the MMC_WriteBoolParameter function.

Remarks

The parameters are defined in the table in section **5.7.14** above for the function block MMC_ReadParameter.

Scope

All

MMC_WRITEBOOLPARAMETER_IN Structure

```
typedef struct{
long lValue;
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_WRITEBOOLPARAMETER_IN;
```

Parameters

lValue

Input value. Any integer.

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_WRITEBOOLPARAMETER_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEBOOLPARAMETER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-68 describes the function block for MMC_WriteBoolParameter

Figure 5-70: MMC_WriteBoolParameter function block

5.7.24.2. Function Block Code Example

```
int rc;
MMC_WRITEBOOLPARAMETER_IN       stWriteBoolParam_in;
MMC_WRITEBOOLPARAMETER_OUT      stWriteBoolParam_out;
//
// Inserting the structure parameters:
stWriteBoolParam_in.lValue          = 1;              // Axis status value of the axis
parameter
stWriteBoolParam_in.eParameterNumber     = MMC_AXIS_STATE_PARAM;// Enumerator value of the
parameter
stWriteBoolParam_in.iParameterArrIndex   = 2;              // Array index parameter
stWriteBoolParam_in.ucEnable             = 1;              // Enabled
//
rc = MMC_WriteBoolParameter (hConn, iAxisRef, &stWriteBoolParam_in, &stWriteBoolParam_out);
if (rc != 0)
{
    HandleError();
}
```

5.7.25. MMC_GlobalWriteBoolParameter

Modifies the value of a vendor global parameter of type BOOL.

```
MMC_LIB_API int MMC_GlobalWriteBoolParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_WRITEBOOLPARAMETER_IN* pInParam,
OUT MMC_WRITEBOOLPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_WRITEBOOLPARAMETER_IN** input data structure using the MMC_GlobalWriteBoolParameter function.

pOutParam

Points to the **MMC_WRITEBOOLPARAMETER_OUT** output structure receiving information, as a result of calling the MMC_GlobalWriteBoolParameter function.

Remarks

The parameters are defined in the table in section **5.7.14** above for the function block MMC_ReadParameter.

Scope

All

MMC_WRITEBOOLPARAMETER_IN Structure

```
typedef struct{
long lValue;
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_WRITEBOOLPARAMETER_IN;
```

Parameters

lValue

Input parameter. Any integer value.

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_WRITEBOOLPARAMETER_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEBOOLPARAMETER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-69 describes the function block for MMC_GlobalWriteBoolParameter

Figure 5-71: MMC_GlobalWriteBoolParameter function block

5.7.25.2. Function Block Code Example

```
int rc;
MMC_WRITEBOOLPARAMETER_IN       stWriteBoolParam_in;
MMC_WRITEBOOLPARAMETER_OUT      stWriteBoolParam_out;
//
// Inserting the structure parameters:
stWriteBoolParam_in.lValue          = 1;              // Axis status value of the axis
parameter
stWriteBoolParam_in.eParameterNumber     = MMC_MOVEMENT_TYPE_PARAM;// Enumerator value of the
parameter
stWriteBoolParam_in.iParameterArrIndex   = 2;              // Array index parameter
stWriteBoolParam_in.ucEnable             = 1;              // Enabled
//
rc = MMC_WriteBoolParameter (hConn, iAxisRef, &stWriteBoolParam_in, &stWriteBoolParam_out);
if (rc != 0)
{
    HandleError();
}
```

5.7.26. MMC_WriteDigitalOutputs

Writes a value to the digital output referenced by the single argument Output (with rising edge of Execute).

```
MMC_LIB_API int MMC_WriteDigitalOutputs(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_WRITEDIGITALOUTPUT_IN* pInParam,
OUT MMC_WRITEDIGITALOUTPUT_OUT* pOutParam
);
```

Motion Mode NC - Supported Distributed - Supported

Source GMAS\includes\MMC_PLCopen_single_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEDIGITALOUTPUT_IN** input data structure using the MMC_WriteDigitalOutputs function.

pOutParam

Points to the **MMC_WRITEDIGITALOUTPUT_OUT** output structure receiving information, as a result of calling the MMC_WriteDigitalOutputs function.

Remarks

This function must be supported for digital output.

Scope

All

MMC_WRITEDIGITALOUTPUT_IN Structure

```
typedef struct MMC_WRITEDIGITALOUTPUT_IN{
int iOutputNumber;
unsigned char ucEnable;
unsigned char ucValue;
}MMC_WRITEDIGITALOUTPUT_IN;
```

Parameters

iOutputNumber

Selects the output. Can be part of MC_OutputRef, if only one single input is referenced. +ve integer value.

Note: **This parameter is not in use at this time.**

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

ucValue

Whether the selected value input signal is True. Boolean values of TRUE/FALSE.

MMC_WRITEDIGITALOUTPUT_OUT Structure

```
typedef struct MMC_WRITEDIGITALOUTPUT_OUT{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEDIGITALOUTPUT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-70 describes the function block for MMC_WriteigitalOutput

Figure 5-72: MMC_WriteDigitalOutput function block

5.7.26.2. Function Block Code Example

```
int rc;
MMC_WRITEDIGITALOUTPUT_IN       stWriteDigOutput_in;
MMC_WRITEDIGITALOUTPUT_OUT      stWriteDigOutput_out;
//
// Inserting the structure parameters:
stWriteDigOutput_in.iOutputNumber   = 16;    //Selects the output depending on the drive
stWriteDigOutput_in.ucEnable        = 1;     //Enabled
//
rc = MMC_WriteDigitalOutputs (hConn, iAxisRef, &stWriteDigOutput_in, &stWriteDigOutput_out);
if (rc != 0)
{
    HandleError();
}
```

5.7.27. MMC_WriteDigitalOutputs32Bit

Writes a value to the 32-bit digital output referenced by the single argument Output (with rising edge of Execute).

```
MMC_LIB_API int MMC_WriteDigitalOutputs32Bit(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_WRITEDIGITALOUTPUT32Bit_IN* pInParam,
OUT MMC_WRITEDIGITALOUTPUT32Bit_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEDIGITALOUTPUT32Bit_IN** input data structure using the MMC_WriteDigitalOutputs32Bit function.

pOutParam

Points to the **MMC_WRITEDIGITALOUTPUT32Bit_OUT** output structure receiving information, as a result of calling the MMC_WriteDigitalOutputs32Bit function.

Remarks

This function must be supported for digital output.

Scope

All

MMC_WRITEDIGITALOUTPUT32Bit_IN Structure

```
typedef struct MMC_WRITEDIGITALOUTPUT32BIT_IN{
int iOutputNumber;
unsigned long ulValue;
unsigned char ucEnable;
}MMC_WRITEDIGITALOUTPUT32Bit_IN;
```

Parameters

iOutputNumber

Selects the output. Can be part of MC_OutputRef, if only one single input is referenced. +ve integer value.

Note: **This parameter is not in use at this time.**

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

ulValue

Total value of all the inputs together. 32 bit bitwise +ve numeric value.

MMC_WRITEDIGITALOUTPUT32Bit_OUT Structure

```
typedef struct MMC_WRITEDIGITALOUTPUT32BIT_OUT{
unsigned short usStatus;
unsigned short usErrorID;
}MMC_WRITEDIGITALOUTPUT32Bit_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-71 describes the function block for MMC_WriteDigitalOutputs32Bit

Figure 5-73: MMC_WriteDigitalOutputs32Bit function block

5.7.27.2. Function Block Code Example

```
int rc;
MMC_WRITEDIGITALOUTPUT32Bit_IN      stWriteDigOutput32Bit_in;
MMC_WRITEDIGITALOUTPUT32Bit_OUT     stWriteDigOutput32Bit_out;
//
// Inserting the structure parameters:
stWriteDigOutput32Bit_in.iOutputNumber   = 16;    //Selects the output depending on the drive
stWriteDigOutput32Bit_in.ucEnable        = 1;     //Enabled
//
rc = MMC_WriteDigitalOutputs32Bit (hConn, iAxisRef, &stWriteDigOutput32Bit_in,
&stWriteDigOutput32Bit_out);
if (rc != 0)
{
    HandleError();
}
```

5.7.28. MMC_WriteParameter

Modifies the value of a vendor specific parameter.

```
MMC_LIB_API int MMC_WriteParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_WRITEPARAMETER_IN* pInParam,
OUT MMC_WRITEPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEPARAMETER_IN** input data structure using the MMC_WriteParameter function.

pOutParam

Points to the **MMC_WRITEPARAMETER_OUT** output structure receiving information as a result of calling the MMC_WriteParameter function.

Remarks

Refer to the full explanation in section **5.4.1 Blended Behaviour Mechanism on page 5-71**.

Scope

All

MMC_WRITEPARAMETER_IN Structure

```
typedef struct{
double dbValue;
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_WRITEPARAMETER_IN;
```

Parameters

dbValue

Parameter value. Any double value.

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_WRITEPARAMETER_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEPARAMETER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-72 describes the function block for MMC_WriteParameter

Figure 5-74: MMC_WriteParameter function block

5.7.28.2. Function Block Code Example

```
int rc;
MMC_WRITEPARAMETER_IN      stWriteParam_in;
MMC_WRITEPARAMETER_OUT     stWriteParam_out;
//
// Inserting the structure parameters:
stWriteParam_in.dbValue             = 1000;                //Parameter value
stWriteParam_in.ucEnable            = 1;                   //Enabled
stWriteParam_in.iParameterArrIndex  = 1;                   //Parameter array index
stWriteParam_in.eParameterNumber    = MMC_SET_POSITION_PARAM;//Axis parameter value
//
rc = MMC_WriteParameter (hConn, iAxisRef, &stWriteParam_in, &stWriteParam_out);
if (rc != 0)
{
    HandleError();
}
```

5.7.29. MMC_GlobalWriteParameter

Modifies the value of a vendor global parameter.

```
MMC_LIB_API int MMC_GlobalWriteParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_WRITEPARAMETER_IN* pInParam,
OUT MMC_WRITEPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	
	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_WRITEPARAMETER_IN** input data structure using the MMC_GlobalWriteParameter function.

pOutParam

Points to the **MMC_WRITEPARAMETER_OUT** output structure receiving information as a result of calling the MMC_GlobalWriteParameter function.

Remarks

For a detailed explanation of this function, refer to the section **5.4.1 Blended Behaviour Mechanism on page 5-71**.

Scope

All

MMC_WRITEPARAMETER_IN Structure

```
typedef struct{
double dbValue;
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_WRITEPARAMETER_IN;
```

Parameters

dbValue

Parameter value. Any double (2 bytes)value.

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_WRITEPARAMETER_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEPARAMETER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-73 describes the function block for MMC_GlobalWriteParameter

Figure 5-75: MMC_GlobalWriteParameter function block

5.7.29.2. Function Block Code Example

```
int rc;
MMC_WRITEPARAMETER_IN      stWriteParam_in;
MMC_WRITEPARAMETER_OUT     stWriteParam_out;
//
// Inserting the structure parameters:
stWriteParam_in.dbValue             = 1000;                //Parameter value
stWriteParam_in.ucEnable            = 1;                   //Enabled
stWriteParam_in.iParameterArrIndex  = 1;                   //Parameter array index
stWriteParam_in.eParameterNumber    = MMC_POSITION_PERIOD_PARAM;//Axis parameter value
//
rc = MMC_GlobalWriteParameter(hConn, iAxisRef, &stWriteParam_in, &stWriteParam_out);
if (rc != 0)
{
    HandleError();
}
```

5.7.30. MMC_AxisLink

This function creates a link between two axes defined as master (Primary) and slave (Minor).

```
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
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_AXISLINK_IN** input data structure using the MMC_AxisLink function.

pOutParam

Points to the **MMC_AXISLINK_OUT** output structure receiving information as a result of calling the MMC_AxisLink function.

Remarks

Linking two axes allows an object to move towards the approximate position of a target and then adjust the movement to accurately pinpoint the target. The initial movement is by the master (Primary) axis, whereas the adjustment is performed by the slave (Minor) axis. The resultant is sum of the master axis's position command, plus the slave axis's position command or the slave axis's actual position.

Scope

All

MMC_AXISLINK_IN Structure

```
typedef struct mmc_axislink_in{
unsigned long ulInputParameter1;
unsigned long ulInputParameter2;
unsigned long ulInputParameter3;
unsigned long ulInputParameter4;
unsigned short usSlaveAxisReference;
unsigned char ucMode;
}MMC_AXISLINK_IN;
```

Parameters

ulInputParameter1

For future use. Present value is 0 or no value.

ulInputParameter2

For future use. Present value is 0 or no value.

ulInputParameter3

For future use. Present value is 0 or no value.

ulInputParameter4

For future use. Present value is 0 or no value.

usSlaveAxisReference

Axis reference of the slave (Minor axis)

ucMode

The position of the master (Primary axis) which can have the following values:

0 – Target Position

1 – Actual position

MMC_AXISLINK_OUT Structure

```
typedef struct mmc_axislink_out{
unsigned short usStatus;
short usErrorID;
}MMC_AXISLINK_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-74 describes the function for MMC_AxisLink.

Figure 5-76: MMC_AxisLink function

5.7.30.2. Function Block Code Example

5.7.31. MMC_AxisUnLink

This function breaks the link between two axes defined as master (Primary) and slave (Minor).

```
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
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_AXISUNLINK_IN** input data structure using the MMC_AxisLink function.

pOutParam

Points to the **MMC_AXISUNLINK_OUT** output structure receiving information as a result of calling the MMC_AxisLink function.

Remarks

None

Scope

All

MMC_AXISUNLINK_IN Structure

```
typedef struct mmc_axisunlink_in{
unsigned char ucdummy;
}MMC_AXISUNLINK_IN;
```

Parameters

ucdummy

Dummy value

MMC_AXISUNLINK_OUT Structure

```
typedef struct mmc_axisunlink_out{
unsigned short usStatus;
short usErrorID;
}MMC_AXISUNLINK_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-75 describes the function for MMC_AxisUnLink.

Figure 5-77: MMC_AxisUnLink function

5.7.31.2. Function Block Code Example

5.7.31.3. Implementation Example

5.8. Multiple Axes Motion Control

Motion Control describes the motion of the multiple axes as either NC or Distributed according to the definitions described in section **2.2 G-MAS Operation Modes** and **2.2.3 G-MAS Axes and Node Definitions** above. The following multiple axes function blocks are described:

Multiple Axes
MMC_GroupStop
MMC_GroupHalt
MMC_MoveCircularAbsolute
MMC_MoveCircularAbsoluteCenter
MMC_MoveCircularAbsoluteBorder
MMC_MoveCircularAbsoluteRadius
MMC_MoveCircularAbsoluteAngle
MMC_MoveLinearAbsolute
MMC_MoveLinearRelative
MMC_MoveLinearAdditive
MMC_MoveLinearAdditiveEx
MMC_MoveLinearAbsoluteRepetitive
MMC_MoveLinearRelativeRepetitive
MMC_MovePath
MMC_PathSelect
MMC_PathUnselect

The structured definitions MMC_MCS_Info_Struct and MMC_MCS_Kin_Ref_Struct are excluded from the above list as they are not function blocks.

5.8.1. Coordinate System and kinematic transformation

The essence of a trajectory is the coordinated motion of one or more axes from a starting point to a target point via a defined path with specified kinematic properties. As for path, one can think of a straight line, a circular movement, or via an input data array. The execution of a path– or any position information - in space requires a coordinate system definition. Within this specification, two coordinate systems are defined; ACS, MCS.

Figure 5-78: Overview of the coordinate systems and transformations

5.8.1.1. Coordinated Motion

When a function block is inserted for operation to a group, the kinematic mode is defined. However, each function block can be inserted in either ACS or MCS kinematic mode. When operating in ACS mode no transformation occurs, but to operate in MCS mode, the kinematic system must be predefined using MMC_SetKinTransform and a transformation applied. **Each kinematic is defined per specific vector**.

5.8.1.2. ACS - Axes Coordinate System

ACS represents the actual position of the physical axis. **ACS** is the mode where all group member kinematic values are downloaded to the axes, without being transformed, or manipulated. Each axis in a group moves as a separate axis (point to point), but all axes' movement start and ends simultaneously. Only linear movements are permitted, but not transitions, and settings for transitions will produce an error. The kinematic limitations refer to the group parameters:

- Max Vector Velocity
- Max Vector AC, DC
- Max Vector Jerk

However, the position values downloaded to the target drive are real values, and are the actual encoder values.

5.8.1.3. MCS - Machine Coordinate System

In this mode, the machine coordinated system is manipulated to allow transformation of kinematic values between the G-MAS and the Drive. All the axes are grouped (to a maximum of 16 axes) to some specific coordinate system. This may be:

Cartesian The Cartesian coordinate system defines the origin as a fixed position relative to the machine. There are three inputs in this mode; X position, Y position, and Z position. For some machines built when using the Cartesian system, the MCS may be identical to ACS, or mapped via a trivial transformation. The coordinate system from the physical multiple axes ACS is linked to the MCS via a kinematic transformation (forward and backward conversion).

Polar For the Polar mode, there are three inputs in this mode; U angle, V angle, and W angle.

Currently this mode is not supported.

N Axes Stand alone axes (N axes). In the N Axes mode, there are nine inputs; N1 – N9 positions.

S Direction The input for S direction **during motion** can be obtained from another kinematic which is currently used.

The **S Direction** may be one of the following

- **Cartesian**
  When using only Cartesian kinematic, the input is spatial path vector (without direction).

$$\Delta Pos_s = \sqrt{\Delta Pos_X{}^2 + \Delta Pos_Y{}^2 + \Delta Pos_Z{}^2}$$

S movement (start from "0")

2D cartesian movement

- **N axes**
  When using only N axes kinematic, the input is a position vector size of all active N axes (without direction)

$$\Delta Pos_s = \sqrt{\Delta Pos_{N1}{}^2 + \Delta Pos_{N2}{}^2 + \ldots + \Delta Pos_{N8}{}^2 + \Delta Pos_{N9}{}^2}$$

- **Cartesian and N axes**
  Act like in Cartesian case

Another method to set the S direction, is using a Parameter mechanism using the **MMC_WriteParameter** function with the MMC_MCS_S_DIRECTION enumerator. This can be **applied for all PLC states**. In contrast, the velocity of the S direction is always non-negative.

In MCS mode, any vector movements are permitted, e.g.linear, circular, spline, etc., but all axes movements start and end simultaneously (in terms of time). The kinematic limitations refer to the parameters per group:

- Max Vector Velocity
- Max Vector AC, DC
- Max Vector Jerk

When the coordinate system is mixed with Cartesian and N axes, only the blended motion is permitted, on the condition that the N axes are not moving, as shown in the table below.

No.	N axes moving	Cartesian axes moving	Transition acceptable
1	YES	YES	X
2	YES	NO	X
3	NO	YES	√

The position values are downloaded to the target drive after a transformation defined by the MMC_SetKinTransform function as described in Figure 5-77 (objects 0X607A, 0X6064, are based on the DS402 protocol).

Figure 5-79: Position values transformed

5.8.1.4. Using the function MMC_SetKinTransform

The function definitions are only relevant for MCS mode, and can only be operated when the group is in non-motion state. The function defines the following parameters:

Parameter	Explanation
Number of axes to be used in motion (*iNumAxes*)	This number defines how many entries to check when reading the other input vectors (node_handlers, axis_type, trans_func, etc…), and should be limited by the total number of axes in the group. It can be smaller than the total number in the group (currently not supported)
Refeence Array	Assign each axis to a specific entry in the array as shown below.
Type of each axis (Axis Reference vs Type)	
Axis Type (*eType*)	This is the kinematic direction. The Type of axis are: • Cartesian types - X, Y, Z • Polar types - U, V, W (not supported) • N axes types – N1, N2, N3, N4, N5, N6, N7, N8, N9 • S type A single Cartesian Direction cannot be selected, and if necessary, revert the axis to ACS modes or an N axis. However, it is possible to set a number of axes with the same type. When using the N axes type, make sure that they are defined consecutively, but the kinematic definition order does not have to be the same order as the N axes definition. It is therefore possible to define the kinematic order; N3, N2, N1 as long as there are no omittions of N axes. Example: Good: (N1,N2,N3), (N1,N2,N1), (N1,N3,N2) Bad: (N1,N5,N6), (N2,N3,N4), (N3,N2,N3)
Axis Reference Array (*iMcsToAcsFuncID*)	Assignment example (groups with 16 axes):

Parameter	Explanation
Axis Reference *(hNode)*	The Node axis reference. Refer to the explanation in section **5.9.12 MMC_SetKinTransform** for this parameter.
Type of transformation with relevant inputs for each axis (*ulTrCoef*)	The transformation is between ACS and MCS Coordinate systems. For each transformation, there are three "double" inputs. Currently Elmo supports only two transition types: • Without Transformation • Linear Transformation **Linear Transformation** The Inputs coefficients should be: • Forward Ratio (FR) • Backward Ratio (BR) • Shift (SH) The Transition functions are defined as: Forward: Position = Position * FR + SH Backward: Position = Position * BD – SH **Without Transformation** The Inputs are not in use: The Transition functions are defined as: Forward: Position = Position Backward: Position = Position Assignment example (group with 16 axes): An important point when entering the coefficients: In order to avoid problematic position adjustments (when using MCS mode) always maintain the following ratio between the coefficients: NC_BACK_TR_RATIO_COEF.NC_BACK_SHIFT_COEF =1 [NC_MAX_NUM_COEF] is defined as 3.

5.8.1.5. Implementation

Each Linear function has an MC_COORD_SYSTEM_ENUM input which define the coordinate system of the current function block:

- ACS
- MCS

The user can change the Coordinate system every function block, but the Group should be in non-motion state.

Each Group function has Position array. The Position input array is understood different in ACS compared to MCS.

Position Array – ACS (each number is indexed inside group)

N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	N13	N14	N15	N16

Position Array – MCS

X	Y	Z	U	V	U	N1	N2	N3	N4	N5	N6	N7	N8	N9	S

5.8.1.6. Example

ACS – Two FB's (Move Linear), Start position – all axes in 0.

FB1 – Position[10,20,30,40,50,60,70,80,90,100,110,120,0,0,0,0]

FB2 – Position[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

MCS - Two FB's (Move Linear), Kinematic: (2-X),(1-Y),(2-Z),(3-N1),(4-N2), Start position – all axes in 0.

FB1 – Position[10, 20, 30, 0, 0, 0, 40, 50, 0, 0, 0, 0, 0, 0, 0, 0]

FB2 – Position[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

These sections describe the special functions and their parameters which perform a transition or circular motion of a single or group axes.

5.8.1.7. Transition Modes

The basic transition modes are defined as follows. Other modes as well as supplier specific modes can be added.

No.	MC_TransitionMode	Description
0	MC_TM_NONE_MODE	Insert no transition curve (default mode)
1	MC_TM_MAX_VELOCITY_MODE	Transition with given start velocity. This is not supported at this time.
2	MC_TM_DEFINED_VELOCITY_MODE	Transition with given constant velocity
3	MC_TM_CORNER_DISTANCE_MODE	Transition with given corner distance
4	MC_TM_MAX_CORNER_DEVIATION_MODE	Transition with given maximum corner deviation
5	MC_TM_SWITCH_RADIUS_MODE	Transition with given radius of the circle arc
7	MC_TM_CORNER_DIST_CV_POLYNOM3	Refer to to the sections 5.8.2.2 - 5.8.2.5 below, for a detailed explanation.
8	MC_TM_CORNER_DIST_CV_POLYNOM5	Refer to to the sections 5.8.2.2 - 5.8.2.5 below, for a detailed explanation.
9	MC_TM_CORNER_DEVIATION_MODE_PLN6	Refer to to the sections 5.8.2.2 - 5.8.2.5 below, for a detailed explanation.

10	MC_TM_CORNER_DIST_CV_POLYNOM5_NAXES	Refer to to the sections 5.8.2.2 - 5.8.2.5 below, for a detailed explanation.
11--…	Supplier specific modes	

5.8.1.8. Single Axis Buffer Modes

These transitions can only be applied in NC mode with Blending in a forward direction. The Input parameters are set using a Buffer mode, but no blending occurs when the function block is presently active.

Blended

Not - Blended

5.8.1.9. Multi-Axes Transitions

Transitions for the multi-axes are only permissible for the Cartesian axis type, but not for reverse geometry (180°). The Input is set in Buffer mode, and the Transition mode uses the appropriate special Transition parameter. When using a special option for polynomial transitions, the “S” parameter is used.

Similar to the Single axis transitions, no blending is allowed with transitions when the function block is active. However, in Multi-axes transitions, the geometry of the transition curve is defined by the Transition mode. The geometric transition involves the use of another function block whose kinematics (velocity) is defined by a Buffer mode, and as a pre-condition, the use of MMC_SetKinTransform.

5.8.1.10. Matrix of available Transition Modes

The matrix in the table below describes the available transition modes for the different buffer modes, and can be used by the supplier to document its supported transition modes.

The matrix of the available transition modes applies to the following transitions:

Line-line

Line-circle

Circle-line

Circle-circle

If two adjacent shapes do not belong to the same plane, or they belong to the plane that is not parallel to a coordinate plane, then only transition modes 7 and 8 are are supported.

TransitionMode > All BufferModes	0	1	2	3	4	5	7	8
Aborting	A	N	N	N	N	N	N	N
Buffered	A	N	N	N	N	A	N	N
Blending Low	N	N	A	A	A	A	A	A
Blending Previous	N	N	A	A	A	A	A	A
Blending Next	N	N	A	A	A	A	A	A
Blending High	N	N	A	A	A	A	A	A

Legend:

Transition Modes 0 – 8 Refer to the table in section **0 Transition Modes**.

A = Available, N = Not supported

The Input parameter Transition parameter is a double data type with various connotations for each transition type. When buffered or in Abortion mode, the value of this parameter is 0. It is also limited by 45% of the shortest segment between two connected consecutive function blocks.

When **Velocity** is defined, the Transition Parameter (TP) is measured in % of the function block's defined velocity. The radius of the Arc is calculated using speed and acceleration according to:

$$R = V^2/Acc$$

Where **V** is the velocity defined by the Blended mode x % of TP

And **Acc** is the maximum vector acceleration, defind by **MAX_AC on page 5-68**.

When the **Switch Radius** is defined, the TP defines the radius of the Arc that connects the two function blocks as shown in **Figure 5-78**.

Figure 5-80: Transition Parameter defined for Switch Radius

When the **Corner Deviation /Polynom6** is defined, the TP defines the minimum distance between the Arc and the buffered transition point as shown in **Figure 5-79** (End point of FB1 and Start point of FB2 in case of Buffered mode).

Figure 5-81: Transition Parameter defined for Corner Deviation/Polynom6

When the **Corner distance\Polynom3\Polynom5** is defined, the TP defines the trajectory distance between the start of the Arc and the transition point (buffered point) as shown in **Figure 5-80** (End point of FB1 and Start point of FB2 in case of Buffered mode).

Figure 5-82: Transition Parameter defined for Corner distance\Polynom3\Polynom5

5.8.1.10.a Transition in 2D and 3D

In 2D space all transition modes 1-8 can be used. In 3D space modes 1- 5 can be used only in the situations where two neighboring shapes are optionally:

- Belong to some coordinate plane XY, XZ, YZ
- Belong to some plane parallel to the coordinate plane

This means that one of the coordinates is constant for all the points of both shapes. If this coordinate constant is zero, both shapes belong to the coordinate plane. However, if not equal to zero, both shapes belong to some plane parallel to the coordinate plane. If this requirement is not fulfilled for the transition in 3D space, modes 7-8 must be used:

- MC_TM_CORNER_DIST_CV_POLYNOM3 = 7
- MC_TM_CORNER_DIST_CV_POLYNOM5 = 8
- (or MC_TM_NONE_MODE = 0 – transition wth zero velocity)

5.8.1.10.b Transition parameter limitation

No matter what transition mode is finally used, it predefines some corner distance. To avoid transition curves intersecting, the G-MAS limits the corner distance to 40% of the original segment length for the transition modes 1-5 and to 45% for the modes 6 - 8.

5.8.1.10.c Transition mode MC_TM_CORNER_DIST_CV_POLYNOM3 = 7

Note: **The Transition parameter must be defined as a corner distance, where the transition is executed with the constant velocity**.

This mode describes a transition curve built as a cubic polynomial of some parameter "S". Initial estimation of the parameter variation is based on the distance between start and end points with the objective to approach the desired curve length.

The constant velocity is originally defined by the blend mode but can be decreased if cannot be achieved due to trajectory parameters. It can be also decreased before the transition curve if the predefined velocity causes at some points of the transition curve 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 the transition is 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 desirable, the user can decrease the value of the parameter, causing a decrease of the transition velocity by the formula

$V = [MMC_MAX_ACCELERATION_PARAM * \rho_{min}]^{1/2}$

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.

5.8.1.10.d Transition mode MC_TM_CORNER_DIST_CV_POLYNOM5 = 8

Note: **The Transition parameter must be defined as a corner distance, where the Transition is executed with the constant velocity.**

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.

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 desirable, the user can decrease the value of the max acceleration parameter causing a decrease of the transition velocity by the formula

$V = [MMC_MAX_ACCELERATION_PARAM * \rho_{min}]^{1/2}$

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.

5.8.1.10.e Transition mode MC_TM_CORNER_DEVIATION_MODE_PLN6 = 9

Note: **The Transition parameter must be defined as a corner deviation, where the maximum distance between the intersection point and the nearest point belongs to the transition curve.**

This mode transition curve is constructed as a sextic polynomial of some parameter "S". Initial estimation of the parameter variation is based on the distance between start and end points with the aim to approach the desirable curve length.

The Transition is executed with constant velocity. This velocity is initially defined by the blend mode but 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.

In this mode, the user also defines the corner distance with the use of the parameter MMC_S_FACTOR. The Corner distance is calculated as
(1 + MMC_S_FACTOR)*TRANSITION_PARAM

where TRANSITION_PARAM defined in the function block, must be equal to the required corner deviation. These two parameters must be reasonably coordinated to achieve desired geometry. In some cases, this may require experimentation.

This transition mode guarantees a transition smoothed by the velocity and acceleration. If, at some points, the acceleration achieved is less than one predefined by the parameter MMC_MAX_ACCELERATION_PARAM but greater than desirable, the user can decrease the value of the maximum acceleration parameter causing a decrease of the transition velocity, using the formula
$V = [MMC_MAX_ACCELERATION_PARAM * \rho_{min}]^{1/2}$
where ρ_{min} is the minimal radius of curvature of 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. It should be noted that this is the only possible mode for the corner deviation in case of a transition between two shapes that do not belong to the same coordinate plane or some plane parallel to coordinate planes XY, XZ, or YZ.

On the other hand, transition mode MC_TM_MAX_CORNER_DEVIATION_MODE = 4 can only be used for two shapes belonging to the same coordinate plane or any two lines intersection in 3D.

5.8.1.11. Transition mode MC_TM_CORNER_DIST_CV_POLYNOM5_NAXES = 10

Until now, in order to perform blended motions, they had to be part of an X-Y-Z Cartesian system, and the user had to work in MCS, and select the Kinematic direction for each axis. However, systems with no Kinematic direction, (as we call – N axes) no blended motion could be performed, and all motions had to be buffered.

A new transition type was introduced; MC_TM_CORNER_DIST_CV_POLYNOM5_NAXES = 10, where this mode is a Corner Distance, fifth order polynomial transition mode for ACS. This mode differs from the mode MC_TM_CORNER_DIST_CV_POLYNOM5 in supporting the N-dimensional smoothed transition. At the moment N is limited by 16.

5.8.1.12. Polynomial transition curve vs circle arc

Transition by the circle arc produces the most natural geometry but has two critical drawbacks:

- Such transition is only possible when two neighbor shapes belong to the same plane.
- Acceleration discontinuity. At start point, there is an acceleration jump V^2/R (where R is the radius of the circle) when moving along the line with constant velocity. The same jump can occur at the end-point of the switch arc.

The disadvantage of the polynomial is a less predictable geometry. In the cases of modes 7 and 8, they can be influenced by the parameter MMC_S_FACTOR. When this parameter increases, the curve length increases and approaches the initial intersection geometry, causing a small radius of curvature and high acceleration. When this parameter decreases, we approach the line connecting the start and end-points of the transition curve, with the resulting low acceleration. The default value 1.4 produces a switch radius slightly smaller than a circle arc. The user can define some value of MMC_S_FACTOR different from the default if the original transition curve does not satisfy his requirements.

Therefore, in the case of a polynomial, we can use two parameters to define the intersection geometry TRANSITION_PARAMETER, which in this polynomial situation refers to the corner distance, and

MMC_S_FACTOR that influences the length of the transition curve. In case of transition by the circle arc, the TRANSITION_PARAMETER (corner distance, corner deviation, switch radius) uniquely defines the intersection geometry.

5.8.1.13. The "S" Parameter

The "S" parameter is a G-MAS group related parameter which is only relevant for polynomial transitions. The Transition length is proportional to the "S" Parameter (CurveLength ~ S) with a range [0…2]. The optimal "S" parameter causes a minimum AC\DC on transition, does not cross the buffering point, and varies according to the angle at the connection buffered point.

Two lines (90 degrees between them) – different S parameter

S = 0.5 **S = 1.2** **S = 1.8**

Two lines (45 degrees between them) – different S parameter

S = 0.5 S = 1.2 S = 1.8

5.8.1.14. Transition Inputs

The following table displays an application of the Buffered mode with the Transition mode.

Buffer mode	Transition Mode application
Aborted Buffered	None
Blending Low Blending Previous Blending Next Blending High	Defined velocity Switch Radius Corner deviation Corner deviation (Polynom 6) Corner distance Corner distance (Polynom 3) Corner distance (Polynom 5)

The MC_BufferMode defines the velocity at the transition point. When **Aborting** is applied:

- All existing function blocks in the node queue are aborted
- The Transition velocity changes to zero, which in turn stops all current motion, and executes the new motion from zero velocity

When **Buffering** is applied, the current function block is inserted to the function block queue in the G-MAS, and on completion of the axis\group execution of all function blocks in the queue (with the except of the current

function block), then the current function block will start its execution. In addition, the Transition velocity is zero.

When **Blending Low** is applied, the current function block is inserted to the function block queue in the G-MAS, and on completion of the axis\group execution of all function blocks in the queue (with the except of the current function block), then the current function block will start its execution. However, the Transition velocity is the smaller value between the previous and current function blocks.

When **Blending Previous** is applied, the current function block is inserted to the function block queue in the G-MAS, and on completion of the axis\group execution of all function blocks in the queue (with the except of the current function block), then the current function block will start its execution. However, the Transition velocity is the velocity of the previous function block.

When **Blending Next** is applied, the current function block is inserted to the function block queue in the G-MAS, and on completion of the axis\group execution of all function blocks in the queue (with the except of the current function block), then the current function block will start its execution. However, the Transition velocity is the velocity of the current function block.

When **Blending High** is applied, the current function block is inserted to the function block queue in the G-MAS, and on completion of the axis\group execution of all function blocks in the queue (with the except of the current function block), then the current function block will start its execution. However, the Transition velocity is the higher between the previous and the current function blocks.

Example (Single Axis)

FB1 – velocity = 100K

FB2 – velocity = 200K

Blending High

Blending Previous

Example (Multi Axis)

FB1 – velocity = 100K

FB2 – velocity = 150K

Blending Next

Blending Low

5.8.1.15. Selecting Polynomial or Simple Transition

The following table describes the advantages and disadvantages of each transition mode.

Transition	Advantage	Disadvantage
Simple	Smooth Position Strongly defined geometry (circle shape) Low AC\DC on transition	Uneven velocity on transition point Uneven AC\DC (current) on transition point
Corner distance Polynom3 – third order	Smooth Position Smooth Velocity	Uneven AC\DC (current) on transition point. Vaguely defined geometry (S parameter), shape limit. High AC\DC on transition
Corner distance Polynom5 – fifth order	Smooth Position Smooth Velocity Smooth AC\DC (current)	Vaguely defined geometry (S parameter), shape limit. Very high AC\DC on transition
Corner deviation Polynom6 – sixth order	Smooth Position Smooth Velocity Smooth AC\DC (current) The deviation point is the closest points to buffered point	Vaguely defined geometry (S parameter) start point of transition. Very high AC\DC on transition.

5.8.1.16. 2D and 3D Transition Examples

The following is a 2D simple example, containing two lines with 90degrees between them:

The sequence is:

1. Start from 0,0
2. Go to 0,1M
3. Go to 1M,1M
4. Go to 1M,0 (Blended)
5. Go to 0,0

This sequence performed three times.

- First time with Transition mode “3”.
- Second time with Transition mode “7”.
- Third time with Transition mode “8”.

NOTE: **All the sequences are performed with the same transition parameter (S=1.4).**

By viewing the position plots alone, it is not possible to indicate any significant difference between them. It is only possible to say that at different corner distance mode (simple, Polynom 3 and 5) the shape of the curve varies.

The following is a complarison between three optional methods to apply the corner distance transition modes, and involves a transition between two linear X,Y motions on the plane.

To view and understand the real difference between transition modes simple, Polynom 3, and 5, it is necessary to look at the Velocity and AC\DC graphs. However, it is necessary to define the differences between them in mathematical terms. The keyword of the differences is “continuity”, as the transition mode increases, the continuity is “better” (higher order).

In Transition mode 3 (Simple), the continuity is only in position. In all cases where the continuity is in position, there are no unexpected jumps or sharp transitions.

In Transition mode 7(Polynom 3), the continuity is in position and velocity.

In Transition mode 8 (Polynom 5), the continuity is in position velocity and AC\DC.

Looking at the graphs for each Transition mode (Position, Velocity and AC\DC), the red circles indicate the points of NON-continuity.

5.8.1.17. Circular Modes

CircMode = BORDER

Defines the end point (*dEndPoint*) and a border point (*dBorderPoint*) on the sector of the circle travelled by the machine (equivalent to the AuxPoint in the PLCopen standard). The border point usually can be reached by the machine, i.e. it can be taught. However, this parameter is restricted to angles < 2π in one single command.

CircMode = CENTER

Defines the endpoint and the center point (*dCenterPoint*) of the circle (equivalent to the AuxPoint in the PLCopen standard). When using this mode, the input *PathChoice* defines whether the Short or the Long sector is travelled by the machine.

This parameter is restricted to angles < 2π and ≠ π in one single command.

CircMode = RADIUS

Defines the end point and the perpendicular vector of the circle plane. The length of the vector corresponds to the radius of the circle. The spearhead point of the vector is the input signal *dSpearHeadPoint* in absolute coordinates (equivalent to the AuxPoint in the PLCopen standard), i.e. referring to the origin of the coordinate system specified in CoordSystem.

If the diameter is larger than the distance between starting and end point, two different circles have to be considered. When using this mode, the user defines a choice of *Pathchoice*; clockwise or counterclockwise, and *ArcShortLong*; Short or Long segment. Clockwise or counterclockwise direction is defined as seen from the end of the perpendicular vector described by the user.

This parameter is restricted to angles < 2π in one single command and the perpendicular vector has to be computed.

Example:

dSpearHeadPoint = (50,0,0) → Circle in plane parallel to yz plane with radius 50 and rotation around axis parallel to x-axis (CoordSystem = MCS).

5.8.1.18. Angle Mode

The angle is the relative angular position for the coordinate system specified by the input signal CoordSystem, measured in degrees.

The angle is defined by:

$$|Angle| > 0$$

Its range may be any +ve or –ve number (e.g. 1012 is quite acceptable). In principle, an angle larger than 360°, means multiple rotations of an object in the direction defined.

5.8.1.19. PathChoice Data verification in MoveCircular functions

Path choice and arc short long parameters data verifications are now added in the MoveCircular functions. These verifications involve a check of the validity of the path choice and arc short long parameter according to the circle mode for the following parameters (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

5.8.2. Move Polynomial Function Block

Polynomial Function Blocks were introduced to improve the ability to build motion sequences with a smooth velocity and AC\DC while performing transitions between motion blocks, for the Linear and Circular motions.

Figure 5-83 Polynomial Function Motion

The motion shape is a 6th order Polynom which crosses three given points:

- Start point
- Auxiliary point
- End point

The Position, Velocity, and AC/DC are controlled to provide the smoothness throughout the path as shown in the following examples.

Figure 5-84 Example of Polynomial Function Motion

Figure 5-85 Example of Polynomial Function Motion within EAS Application

5.8.3. Splines

Splines can be defined for the motion of a vector/system, designed where all the potential trajectory points are known in advance. The user is interested in the smoothest trajectory obtainable. Therefore using Splines may be the solution.

Splines are defined as n dimensional position points (1 < n<=16), which describe the geometry of the trajectory. The kinematics of the trajectory can be defined in two ways:

a) By the time interval between each point

b) By the kinematic parameters: Max Velocity, Max Acceleration, Max Deceleration, and Jerk.

Either way, the spline is designed as a sequence of cubic polynomials continuous by the first and second derivatives. The following Spline modes define kinematic options:

Spline mode	Explanation
MC_SPLINE_MODE_FT = 1	In modes MC_SPLINE_MODE_FT and MC_SPLINE_MODE_V_AC_DC the Spline Velocity is a parabolic function of time.
MC_SPLINE_MODE_V_AC_DC = 2	
MC_SPLINE_MODE_CV = 7	In modes MC_SPLINE_MODE_CV and MC_SPLINE_MODE_NP the Spline is executed with Constant Velocity with an Acceleration segment at the start and Deceleration segment at the end.
MC_SPLINE_MODE_NP = 3	

When would a user prefer one mode to another?

If the user requires a minimal time for the whole trajectory but constant velocity is not required, the Spline mode V_AC_DC should be selected.

If on the other hand, the user knows the precise time intervals between each two points and a constant velocity is not required, the Spline mode FT should be selected.

If the trajectory must be executed with constant velocity, then the user should select the Spline modes CV or NP.

Having selected the appropriate Spline mode, a Spline data table (or series of Spline tables) is created (in Excel or otherwise) according to the definitions and explanations described in the next sub-sections. This Spline table is saved as a text file, and downloaded to the G-MAS. Its possible file path may then be, for example: /mnt/jffs/usr/Spline_2D.txt

5.8.3.1. Spline Mode MC_SPLINE_MODE_FT = 1

In spline mode FT the user defines the spline mode, dimension (number of axes 1-16), number of points N and a table of points. Each row of the table has the same structure and contains information for the point i. The first column of the row i contains dT_i, the time interval (in seconds) required to arrive at point i from point i-1. The next row contains the position coordinates for the point i.

The example below is part of the spline table (first five points) for the butterfly spline.

Spline data			
Spline Mode:			1
Spline Dimension:			2
Spline number of points:			433
Spline data start			
	2.000000	35914.091423	0.000000
	2.000000	56390.497961	9943.166246
	2.000000	103925.224490	37825.688304
	2.000000	146247.550991	84436.062933
	2.000000	154381.879874	129541.778462

Spline in mode FT comprises of a function of time in continuous velocity and acceleration. Each segment i is a cubic polynomial of time with parameter deviation dT_i. This polynomial is built for each coordinate j. Velocities and accelerations are predefined by the parameters dT_i, i = 1,2,….N, but are limited by the parameters MMC_MAX_VELOCITY_PARAM and MMC_MAX_ACCELERATION_PARAM. Therefore, the initial values of dT_i can be increased to satisfy these constraints.

For the user to decrease the max. velocity or acceleration on some segment i, he can increase dT_i. To prevent undesirable deviations from the original polygon, some position control is implemented for the short splines (N < 50). This is performed by inserting additional internal points. If the user is dissatisfied with some interval on the spline curve, he can manually add a number of internal points.

Spline mode MC_SPLINE_MODE_FT should be used if the user can define precise time intervals for each segment and is not interested in constant velocity along the spline.

5.8.3.2. Spline mode MC_SPLINE_MODE_V_AC_DC = 2

In the spline mode MC_SPLINE_MODE_V_AC_DC the user defines the following:

- Spline mode
- Dimension(number of axes 1-16)
- Number of points N, max. velocity
- Max. acceleration (AC)
- Max. deceleration (DC)

- Max. jerk (or alternatively smooth factors for AC and DC)
- Table of points

Each table row has the same structure and contains the data for the point i. All columns contain position coordinates for the point i (1-16).

The example below is part of the spline table (first five points) for the spline.

Spline data	
Lissajous	
Spline mode:	2
Spline dimension:	2
Spline number of points:	145
Max Velocity:	32000.000000
Max AC:	1000000.000000
Max DC:	1000000.000000
Max Jerk:	20000000.000000
SF AC:	0.000000
SF DC:	0.000000
Splines data start	
50000.000000	0.000000
49572.243069	10821.980697
48296.291314	21130.913087
46193.976626	30438.071450
43301.270189	38302.222156

Spline in mode V/AC/DC comprises of a function of time in continuous velocity and acceleration. Each segment i is a cubic polynomial of time, specifically built for each coordinate j (1-16). Velocities and accelerations are limited by user defined parameters.

To prevent undesirable deviations from the original polygon, some position control is implemented for the short splines (N < 50). This is performed by inserting additional internal points. If the user is dissatisfied by the some interval of the spline curve, he can manually add a number of internal points.

Spline mode MC_SPLINE_MODE_V_AC_DC should be used if the user cannot define precise time intervals for each segment and is not interested in constant velocity along the spline curve but can define limitations on max velocity and AC/DC along the path.

5.8.3.3. Spline mode MC_SPLINE_MODE_NP = 3

In the spline mode MC_SPLINE_MODE_NP the user defines the following:

- Spline mode
- Dimension(number of axes 1-16),
- Number of points
- Max velocity
- Max AC
- Max DC
- Max jerk (or alternatively smooth factors for AC and DC)
- Table of points.

Each table row has the same structure and contains the data for point i. All columns contain position coordinates for the point i (1-16). The example below is part of the spline table (first five points) for the butterfly spline.

Spline data	
Spline mode:	8
Spline dimension:	2
Number of spline points:	433
Max Velocity:	100000.000000
Max AC:	10000000.000000
Max DC:	10000000.000000
Max Jerk:	20000000.000000
SF AC:	0.000000
SF DC:	0.000000
Splines data start	
35914.091423	0.000000
56390.497961	9943.166246
103925.224490	37825.688304
146247.550991	84436.062933
154381.879874	129541.778462

Splines in mode CV comprise of a function of some continuous parameter "**S**". Each segment i is a cubic polynomial of **s,** specifically built for each coordinate j (1-16), with velocities and accelerations limited by user defined parameters. The Spline is executed with a constant velocity that is less or equal to the max velocity defined by the user.

The difference with the mode CV is in definition of the parameter "S", whereas the NP modes is close to the natural parameter. In reality, for short splines (N < 50) and smoothed curves, the parameter is completely natural. The advantage of the natural parameter is that the resultant spline does not require any additional calculations in real-time. On the other hand, to build such a spline without using the natural parameter can employ considerable off-line calculations (a couple of minutes for the long and unsmoothed spline). Smoothed splines like ellipse and helix do not need long iterative calculations.

The Spline is executed as a normal G-MAS trajectory with the acceleration limited by jerk, constant velocity stage, and deceleration.

If the user is dissatisfied with some interval of the spline curve, he can manually add a number of internal points.

The Spline mode MC_SPLINE_MODE_NP should be used if the user is interested in constant velocity along the spline curve. This constant velocity is limited by the user defined V_{max} and
V_ρ = [MMC_MAX_ACCELERATION_PARAM *ρ_{min}]$^{1/2}$
where ρ_{min} is the minimal radius of curvature on the spline curve.

So $V_{const} = \min(V_{max}, V_\rho)$.

This limitation usually does not influence smoothed trajectories like ellipse or helixes. This mode is recommended in situations where ther are problems with the CV mode.

5.8.3.4. Spline mode MC_SPLINE_MODE_CV = 7

In the spline mode MC_SPLINE_MODE_CV the user defines the following:

- Spline mode
- Dimension(number of axes 1-16)
- Number of points
- Max velocity
- Max AC, max DC
- Max jerk (or alternatively smooth factors for AC and DC)
- Table of points

Each table row has the same structure and contains the data for the point i. All columns contain position coordinates for the point i (1-16).

The example below is part of the spline table (first five points) for the butterfly spline.

Spline data	
Spline mode:	7
Spline dimension:	2
Number of spline points:	433
Max Velocity:	100000.000000
Max AC:	10000000.000000

Max DC:	10000000.000000
Max Jerk:	20000000.000000
SF AC:	0.000000
SF DC:	0.000000
Splines data start	
35914.091423	0.000000
56390.497961	9943.166246
103925.224490	37825.688304
146247.550991	84436.062933
154381.879874	129541.778462

Splines in mode CV comprises of a function of some continuous velocity and acceleration dependent parameter "**S**". Each segment i is a cubic polynomial of "**S**", specifically built for each coordinate j (1-16), where the velocities and accelerations are limited by user defined parameters. The Spline is executed with constant velocity that is less or equal to the maximum velocity defined by the user, as a normal G-MAS trajectory with the acceleration limited by the jerk, constant velocity stage and deceleration.

If the user is dissatisfied with the some interval of the spline curve, he can manually add a number of internal points.

The Spline mode MC_SPLINE_MODE_CV should be used where the user is interested in constant velocity along the spline curve. This constant velocity is limited by the user defined
V_{max} and V_{ρ} = [MMC_MAX_ACCELERATION_PARAM *ρ_{min}]$^{1/2}$
where ρ_{min} is the minimal radius of curvature on the spline curve.

So $V_{constant} = \min(V_{max}, V_{\rho})$.

This limitation usually does not influence smoothed trajectories like ellipse or helix.

5.8.3.5. Implementing the Spline Motion

The final Spline table(s) is now downloaded to the G-MAS to reside in a known directory with a specific filepath. The functions associated with Spline are used to access the Spline table data. These are:

- MMC_PathSelect
- MMC_MovePath
- MMC_PathUnselect

The function MMC_PathSelect uses the parameter pPathToSplineFile to select and calculate the trajectory by defining the path to the Spline table. The function will then create an index for each Spline table, in the G-MAS memory. The function MovePath then moves along the trajectory path using the hMemHandle index to indicate the path to move the vector/system.

5.8.4. NC_MCS_Info_Struct

NC_MCS_INFO_STRUCT Structure

```
typedef struct{
double ulTrCoef[NC_MAX_NUM_COEF];
unsigned int hNode;
NC_AXIS_IN_GROUP_TYPE_ENUM eType;
NC_TR_FUNC_ID_ENUM eMcsToAcsFuncID;
}NC_MCS_INFO_STRUCT;
```

Description Info structural parameters. This structure is not a function block.

Source GMAS\includes\MMC_PLCopen_group_API.h

Parameters

ulTrCoef [NC_MAX_NUM_COEF]

Defined by the type of supported MCS transform coefficients according to the definition of the enumerator NC_TR_COEF_TYPE_ENUM:

NC_BACK_TR_RATIO_COEF = 0

NC_FRWD_TR_RATIO_COEF = 1

NC_BACK_SHIFT_COEF = 2

The parameter ulTrCoef can have values of any +ve double values.

[NC_MAX_NUM_COEF] can have array values of [1….n].

hNode

Node index integer. Any positive numeric value.

eType

MC_AXIS_IN_GROUP_TYPE_ENUM define the types of supported axis in the group. The are 11 possible values:

NC_X_AXIS_TYPE = 0
NC_Y_AXIS_TYPE
NC_Z_AXIS_TYPE
NC_U_AXIS_TYPE
NC_V_AXIS_TYPE
NC_W_AXIS_TYPE
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
NC_S_AXIS_TYPE

eMcsToAcsFuncID

MC_TR_FUNC_ID_ENUM can have IDs 0…15 integer values for the 16 Nodes:

NC_NODE_1_ID = 0

……

NC_NODE_16_ID = 15

5.8.5. NC_MCS_Kin_Ref_Struct

NC_MCS_Kin_Ref_Struct Structure

```
typedef struct{
int iNumAxes;
NC_MCS_INFO_STRUCT sMcsInfo[NC_MAX_NUM_AXES_IN_NODE];
}NC_MCS_KIN_REF_STRUCT;
```

Description MCS Info structural parameters. This structure is not a function block.

Source GMAS\includes\MMC_PLCopen_group_API.h

Parameters

iNumAxes

Number of axes in group. Any positive integer.

sMcsInfo [NC_MAX_NUM_AXES_IN_NODE]

Refer to the previous function ***5.8.4 NC_MCS_Info_Struct*** for a detailed explanation of the parameter.

The array size *[NC_MAX_NUM_AXES_IN_NODE]* is equal to 16, as the maximum number of axes in a group. The parameter sMcsInfo is a structure, whereas the array parmeter has values of [2….15].

5.8.6. MMC_GroupStop

For multi-axis systems. Brings a group of axes to stop status.

```
MMC_LIB_API int MMC_GroupStopCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPSTOP_IN* pInParam,
OUT MMC_GROUPSTOP_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPSTOP_IN** group axes input structure that receives the MMC_GroupStop command.

pOutParam

Points to the **MMC_GROUPSTOP_OUT** group axes output structure receiving information, as a result of calling the MMC_GroupStop function.

Remarks

None

Scope

Only MCS coordinate system is supported.

MMC_GROUPSTOP_IN Structure

```
typedef struct MMC_Group_Stop_in{
float fDeceleration;
float fJerk;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
}MMC_GROUPSTOP_IN;
```

Parameters

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_GROUPSTOP_OUT Structure

```
typedef struct MMC_Group_Stop_out{
unsigned int uiHndl;
unsigned short usStatus;
unsigned short usErrorID;
}MMC_GROUPSTOP_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-81 describes the function block for MMC_GroupStop

Figure 5-86: MMC_GroupStop function block

5.8.6.2. Function Block Code Example

```
int rc;
MMC_GROUPSTOP_IN       stGroupStop_in;
MMC_GROUPSTOP_OUT      stGroupStop_out;
//
// Inserting the structure parameters:
stGroupStop_in.fDeceleration    = 100000.0;                //Value of the deceleration
stGroupStop_in.fJerk            = 20000000.0;              //Maximum value of the Jerk
stGroupStop_in.eBufferMode      = MC_BUFFERED_MODE;        //Defines the behavior of the axis
stGroupStop_in.ucExecute        = 1;
//
rc = MMC_GroupStopCmd (hConn, iAxisRef, &stGroupStop_in, &stGroupStop_out);
if (rc != 0)
{
    HandleError();
}
```

5.8.7. MMC_GroupHalt

For multi-axis systems. Brings a group of axes to Halt status.

```
MMC_LIB_API int MMC_GroupHaltCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPHALT_IN* pInParam,
OUT MMC_GROUPHALT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPHALT_IN** group axes input structure that receives the MMC_GroupHalt command.

pOutParam

Points to the **MMC_GROUPHALT_OUT** group axes output structure receiving information, as a result of calling the MMC_GroupHalt function.

Remarks

None

Scope

Only MCS coordinate system is supported.

MMC_GROUPHALT_IN Structure

```
typedef struct MMC_Group_Halt_in{
float fDeceleration;
float fJerk;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
}MMC_GROUPHALT_IN;
```

Parameters

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_GROUPHALT_OUT Structure

```
typedef struct MMC_Group_Halt_out{
unsigned int uiHndl;
unsigned short usStatus;
unsigned short usErrorID;
}MMC_GROUPHALT_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-82 describes the function block for MMC_GroupHalt

Figure 5-87: MMC_GroupHalt function block

5.8.7.2. Function Block Code Example

```
int rc;
MMC_GROUPHALT_IN       stGroupHalt_in;
MMC_GROUPHALT_OUT      stGroupHalt_out;
//
// Inserting the structure parameters:
stGroupHalt_in.fDeceleration    = 100000.0;            //Value of the deceleration
stGroupHalt_in.fJerk            = 20000000.0;          //Maximum value of the Jerk
stGroupHalt_in.eBufferMode      = MC_BUFFERED_MODE;    //Defines the behavior of the axis
stGroupHalt_in.ucExecute        = 1;
//
rc = MMC_GroupHaltCmd (hConn, iAxisRef, &stGroupHalt_in, &stGroupHalt_out);
if (rc != 0)
{
    HandleError();
}
```

5.8.8. MMC_MoveCircularAbsolute

For multi-axis systems. Commands an interpolated circular movement on an axes group from the actual position of the TCP.

```
MMC_LIB_API int MMC_MoveCircularAbsoluteCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVECIRCULARABSOLUTE_IN* pInParam,
OUT MMC_MOVECIRCULARABSOLUTE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVECIRCULARABSOLUTE_IN** group axes input structure that receives the MMC_MoveCircularAbsolute command.

pOutParam

Points to the **MMC_MOVECIRCULARABSOLUTE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveCircularAbsolute function.

Remarks

The end point and auxiliary point, depending on the applied mode, are defined absolutely in the specified coordinate system.

Scope

Only MCS coordinate system is supported.

MMC_MOVECIRCULARABSOLUTE_IN Structure

```
typedef struct{
double dAuxPoint[NC_MAX_NUM_AXES_IN_NODE];
double dEndPoint[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
NC_PATH_CHOICE_ENUM ePathChoice;
NC_ARC_SHORT_LONG_ENUM eArcShortLong;
NC_CIRC_MODE_ENUM eCircleMode;
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucSuperimposed;
unsigned char ucExecute;
}MMC_MOVECIRCULARABSOLUTE_IN;
```

Parameters

dAuxPoint [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute positions for each dimension in the coordinate system specified by the input signal CoordSystem, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dAuxPoint can have vector array [1….3] double values in a technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

dEndPoint [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute end point positions for each dimension in the coordinate system specified by the input signal CoordSystem, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dEndPoint is a 2D or 3D double vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes**.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

ePathChoice

Defines the NC_PATH_CHOICE_ENUM enumerator types of supported path choice. The option are:

MC_NONE_PATH_CHOICE = 0

MC_CLOCKWISE = 1

MC_COUNTERCLOCKWISE = 2

eArcShortLong

Defines the types of supported arc length. The NC_ARC_SHORT_LONG_ENUM enumerator options are:

MC_NONE_ARC_CHOICE = 0

MC_SHORT = 1

MC_LONG = 2

eCircleMode

Defines the types of supported circular modes in 2D. Refer to the section **5.8.2.1 Transition Modes**. The NC_CIRC_MODE_ENUM enumerator options are:

MC_NONE_CIRC_MODE = 0

MC_BORDER_CIRC_MODE= 1

MC_CENTER_CIRC_MODE = 2

MC_RADIUS_CIRC_MODE = 3

MC_ANGLE_CIRC_MODE = 4

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes** and options below. The options are:

MC_TM_NONE_MODE	= 0
MC_TM_MAX_VELOCITY_MODE	= 1, Not supported at this time
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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVECIRCULARABSOLUTE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVECIRCULARABSOLUTE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-83 describes the function block for MMC_MoveCircularAbsolute

Figure 5-88: MMC_MoveCircularAbsolute function block

5.8.8.2. Function Block Code Example

```
int rc;
MMC_MOVECIRCULARABSOLUTE_IN         stMoveCircularAbs_in;
MMC_MOVECIRCULARABSOLUTE_OUT        stMoveCircularAbs_out;
//
// Inserting the structure parameters:
stMoveCircularAbs_in.fAcceleration           = 100000.0;            //Value of the acceleration
stMoveCircularAbs_in.fDeceleration           = 100000.0;            //Value of the deceleration
stMoveCircularAbs_in.fJerk                   = 20000000.0;          //Maximum value of the Jerk
stMoveCircularAbs_in.dAuxPoint[0]            = 5000;  //Absolute center positions for each
dimension
stMoveCircularAbs_in.dAuxPoint[1]            = 5000;  //Absolute center positions for each
dimension
stMoveCircularAbs_in.fTransitionParameter[0] = 10000; //Transition parameters which
characterize the contour curve
stMoveCircularAbs_in.fTransitionParameter[1] = 10000;               //Transition parameters
which characterize the contour curve
stMoveCircularAbs_in.eArcShortLong           = MC_SHORT;   //Defines the types of supported arc
length
stMoveCircularAbs_in.eCoordSystem            = MC_MCS_COORD;    //Supported coordinate system
stMoveCircularAbs_in.eTransitionMode         = MC_TM_DEFINED_VELOCITY_MODE;//Supported
transition modes
stMoveCircularAbs_in.eBufferMode             = MC_BUFFERED_MODE; //Defines the behavior of the
axis
stMoveCircularAbs_in.dEndPoint[0]            = 5000.0; //Absolute end point positions for each
dimension
stMoveCircularAbs_in.dEndPoint[1]            = 5000.0;//Absolute end point positions for each
dimension
stMoveCircularAbs_in.fVelocity               = 5000.0;              //Maximum Velocity of the
path
stMoveCircularAbs_in.ePathChoice             = MC_CLOCKWISE;        //Supported path choice
stMoveCircularAbs_in.eCircleMode             = MC_BORDER_CIRC_MODE; //Supported circular modes
in 2D
stMoveCircularAbs_in.ucSuperimposed          = 1;          //Option to superimpose is operated
stMoveCircularAbs_in.ucExecute               = 1;
//
rc = MMC_MoveCircularAbsoluteCmd (hConn, iAxisRef, &stMoveCircularAbs_in,
&stMoveCircularAbs_out);
```

```
if (rc != 0)
{
    HandleError();
}
```

5.8.8.3. Implementation Example

The example below shows two MMC_MoveCircular Absolute function blocks operation in conjunction. Their timing diagram displays dots on the red line, based on the same timing differences and representing the velocity.

Figure 5-89: MMC_MoveCircularAbsolute - Example

Figure 5-90: MMC_MoveCircularAbsolute timing diagram - Example

Figure 5-91: Timing diagram for MMC_MoveCircularAbsolute – Example

5.8.9. MMC_MoveCircularAbsoluteCenter

For multi-axis systems. Commands an interpolated circular center movement on an axes group from the actual position of the TCP.

```
MMC_LIB_API int MMC_MoveCircularAbsoluteCenterCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVECIRCULARABSOLUTECENTER_IN* pInParam,
OUT MMC_MOVECIRCULARABSOLUTE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVECIRCULARABSOLUTECENTER_IN** group axes input structure that receives the MMC_MoveCircularAbsoluteCenter command.

pOutParam

Points to the **MMC_MOVECIRCULARABSOLUTE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveCircularAbsoluteCenter function.

Remarks

The end point and auxiliary point, depending on the applied mode, are defined absolutely in the specified coordinate system.

Scope

Only MCS coordinate system is supported.

MMC_MOVECIRCULARABSOLUTECENTER_IN Structure

```
typedef struct{
double dCenterPoint[NC_MAX_NUM_AXES_IN_NODE];
double dEndPoint[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
NC_ARC_SHORT_LONG_ENUM eArcShortLong;
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucSuperimposed;
unsigned char ucExecute;
}MMC_MOVECIRCULARABSOLUTECENTER_IN;
```

Parameters

dCenterPoint [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute center positions for each dimension in the coordinate system specified by the input signal CoordSystem, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dCenterPoint can have vector array [1….3] double values in a technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

dEndPoint [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute end point positions for each dimension in the coordinate system specified by the input signal CoordSystem, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dEndPoint is a 2D or 3D double vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes**.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eArcShortLong

Defines the types of supported arc length. The NC_ARC_SHORT_LONG_ENUM enumerator options are:

MC_NONE_ARC_CHOICE = 0

MC_SHORT = 1

MC_LONG = 2

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes** and options below. The options are:

MC_TM_NONE_MODE = 0

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVECIRCULARABSOLUTE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVECIRCULARABSOLUTE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-87 describes the function block for MMC_MoveCircularAbsoluteCenter

Figure 5-92: MMC_MoveCircularAbsoluteCenter function block

5.8.9.2. Function Block Code Example

```
int rc;
MMC_MOVECIRCULARABSOLUTECENTER_IN   stMoveCircularAbsCenter_in;
MMC_MOVECIRCULARABSOLUTE_OUT        stMoveCircularAbs_out;
//
// Inserting the structure parameters:
stMoveCircularAbsCenter_in.fAcceleration              = 100000.0;       //Value of the
acceleration
stMoveCircularAbsCenter_in.fDeceleration              = 100000.0;       //Value of the
deceleration
stMoveCircularAbsCenter_in.fJerk                      = 20000000.0;     //Maximum value of the
Jerk
stMoveCircularAbsCenter_in.dCenterPoint[0]            = 5000;//Absolute center positions for
each dimension
stMoveCircularAbsCenter_in.dCenterPoint[1]            = 5000;//Absolute center positions for
each dimension
stMoveCircularAbsCenter_in.fTransitionParameter[0]    = 10000;          //Transition
parameters which characterize the contour curve
stMoveCircularAbsCenter_in.fTransitionParameter[1]    = 10000;          //Transition
parameters which characterize the contour curve
stMoveCircularAbsCenter_in.eArcShortLong              = MC_SHORT;//Defines the types of
supported arc length
stMoveCircularAbsCenter_in.eCoordSystem               = MC_MCS_COORD; //Supported coordinate
system
stMoveCircularAbsCenter_in.eTransitionMode            = MC_TM_DEFINED_VELOCITY_MODE;
     //Supported transition modes
stMoveCircularAbsCenter_in.eBufferMode                = MC_BUFFERED_MODE;//Defines the
behavior of the axis
stMoveCircularAbsCenter_in.dEndPoint[0]               = 5000.0;     //Absolute end point
positions for each dimension
stMoveCircularAbsCenter_in.dEndPoint[1]               = 5000.0;     //Absolute end point
positions for each dimension
stMoveCircularAbsCenter_in.fVelocity                  = 5000.0;     //Maximum Velocity of the
path
stMoveCircularAbsCenter_in.ucSuperimposed             = 1;          //Option to superimpose is
operated
stMoveCircularAbsCenter_in.ucExecute                  = 1;
//
rc = MMC_MoveCircularAbsoluteCenterCmd (hConn, iAxisRef, &stMoveCircularAbsCenter_in,
&stMoveCircularAbs_out);
if (rc != 0)
{
    HandleError();
}
```

5.8.10. MMC_MoveCircularAbsoluteBorder

For multi-axis systems. Commands an interpolated circular border movement on an axes group from the actual position of the TCP.

```
MMC_LIB_API int MMC_MoveCircularAbsoluteBorderCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVECIRCULARABSOLUTEBORDER_IN* pInParam,
OUT MMC_MOVECIRCULARABSOLUTE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVECIRCULARABSOLUTEBORDER_IN** group axes input structure that receives the MMC_MoveCircularAbsoluteBorder command.

pOutParam

Points to the **MMC_MOVECIRCULARABSOLUTE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveCircularAbsoluteBorder function.

Remarks

The end-point and auxiliary point, depending on the applied mode, are defined absolutely in the specified coordinate system.

Scope

Only MCS coordinate system is supported.

MMC_MOVECIRCULARABSOLUTEBORDER_IN Structure

```
typedef struct{
double dBorderPoint[NC_MAX_NUM_AXES_IN_NODE];
double dEndPoint[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucSuperimposed;
unsigned char ucExecute;
}MMC_MOVECIRCULARABSOLUTEBORDER_IN;
```

Parameters

dBorderPoint [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute border positions for each dimension in the coordinate system specified by the input signal CoordSystem, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dBorderPoint can have vector array [1….3] double values in a technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

dEndPoint [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute end point positions for each dimension in the coordinate system specified by the input signal CoordSystem, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dEndPoint is a 2D or 3D double vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes**.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes** and options below. The options are:

MC_TM_NONE_MODE = 0

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVECIRCULARABSOLUTE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVECIRCULARABSOLUTE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-88 describes the function block for MMC_MoveCircularAbsoluteBorder

Figure 5-93: MMC_MoveCircularAbsoluteBorder function block

5.8.10.2. Function Block Code Example

```
int rc;
MMC_MOVECIRCULARABSOLUTEBORDER_IN   stMoveCircularAbsBorder_in;
MMC_MOVECIRCULARABSOLUTE_OUT        stMoveCircularAbs_out;
//
// Inserting the structure parameters:
stMoveCircularAbsBorder_in.fAcceleration              = 100000.0;       //Value of the
acceleration
stMoveCircularAbsBorder_in.fDeceleration              = 100000.0;       //Value of the
deceleration
stMoveCircularAbsBorder_in.fJerk                      = 20000000.0;     //Maximum value of the
Jerk
stMoveCircularAbsBorder_in.dBorderPoint[0]            = 10000;          //Absolute border
positions for each dimension
stMoveCircularAbsBorder_in.dBorderPoint[1]            = 10000; //Absolute border positions for
each dimension
stMoveCircularAbsBorder_in.fTransitionParameter[0]    = 10000;          //Transition
parameters which characterize the contour curve
stMoveCircularAbsBorder_in.fTransitionParameter[1]    = 10000;          //Transition
parameters which characterize the contour curve
stMoveCircularAbsBorder_in.eCoordSystem               = MC_MCS_COORD;   //Supported coordinate
system
stMoveCircularAbsBorder_in.eTransitionMode            = MC_TM_DEFINED_VELOCITY_MODE;
     //Supported transition modes
stMoveCircularAbsBorder_in.eBufferMode                = MC_BUFFERED_MODE;//Defines the
behavior of the axis
stMoveCircularAbsBorder_in.dEndPoint[0]               = 10000; //Absolute end point positions
for each dimension
stMoveCircularAbsBorder_in.dEndPoint[1]               = 10000;          //Absolute end point
positions for each dimension
stMoveCircularAbsBorder_in.fVelocity                  = 5000.0;         //Maximum Velocity of
the path
stMoveCircularAbsBorder_in.ucSuperimposed             = 1;              //Option to
superimpose is operated
stMoveCircularAbsBorder_in.ucExecute                  = 1;
//
rc = MMC_MoveCircularAbsoluteBorderCmd (hConn, iAxisRef, &stMoveCircularAbsBorder_in,
&stMoveCircularAbs_out);
if (rc != 0)
{
    HandleError();
}
```

5.8.11. MMC_MoveCircularAbsoluteRadius

For multi-axis systems. Commands an interpolated circular movement radius on an axes group from the actual position of the TCP.

```
MMC_LIB_API int MMC_MoveCircularAbsoluteRadiusCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVECIRCULARABSOLUTERADIUS_IN* pInParam,
OUT MMC_MOVECIRCULARABSOLUTE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MOVECIRCULARABSOLUTERADIUS_IN** group axes input structure that receives the MMC_MoveCircularAbsoluteRadius command.

pOutParam

Points to the **MMC_MOVECIRCULARABSOLUTE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveCircularAbsoluteRadius function.

Remarks

The end-point and auxiliary point, depending on the applied mode, are defined absolutely in the specified coordinate system.

Scope

Only MCS coordinate system is supported.

MOVECIRCULARABSOLUTERADIUS_IN Structure

```
typedef struct{
double dSpearHeadPoint[NC_MAX_NUM_AXES_IN_NODE];
double dEndPoint[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
NC_PATH_CHOICE_ENUM ePathChoice;
NC_ARC_SHORT_LONG_ENUM eArcShortLong;
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucSuperimposed;
unsigned char ucExecute;
}MMC_MOVECIRCULARABSOLUTERADIUS_IN;
```

Parameters

dSpearHeadPoint [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute radius positions for each dimension in the coordinate system specified by the input signal CoordSystem, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dSpearHeadPoint can have vector array [1….3] double values in a technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

dEndPoint [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute end point positions for each dimension in the coordinate system specified by the input signal CoordSystem, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dEndPoint is a 2D or 3D double vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes**.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

ePathChoice

Defines the NC_PATH_CHOICE_ENUM enumerator types of supported path choice. The option are:

MC_NONE_PATH_CHOICE = 0

MC_CLOCKWISE = 1

MC_COUNTERCLOCKWISE = 2

eArcShortLong

Defines the types of supported arc length. The NC_ARC_SHORT_LONG_ENUM enumerator options are:

MC_NONE_ARC_CHOICE = 0

MC_SHORT = 1

MC_LONG = 2

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes** and options below. The options are:

MC_TM_NONE_MODE	= 0
MC_TM_MAX_VELOCITY_MODE	= 1, Not supported at this time
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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVECIRCULARABSOLUTE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVECIRCULARABSOLUTE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-89 describes the function block for MMC_MoveCircularAbsoluteRadius.

Figure 5-94: MMC_MoveCircularAbsoluteRadius function block

5.8.11.2. Function Block Code Example

```
int rc;
MMC_MOVECIRCULARABSOLUTERADIUS_IN   stMoveCircularAbsRadius_in;
MMC_MOVECIRCULARABSOLUTE_OUT        stMoveCircularAbs_out;
//
// Inserting the structure parameters:
stMoveCircularAbsRadius_in.fAcceleration              = 100000.0;       //Value of the
acceleration
stMoveCircularAbsRadius_in.fDeceleration              = 100000.0;       //Value of the
deceleration
stMoveCircularAbsRadius_in.fJerk                      = 20000000.0;     //Maximum value of the
Jerk
stMoveCircularAbsRadius_in.dSpearHeadPoint[0]         = 5000;//Absolute radius positions for
each dimension
stMoveCircularAbsRadius_in.dSpearHeadPoint[1]         = 5000;//Absolute radius positions for
each dimension
stMoveCircularAbsRadius_in.fTransitionParameter[0]    = 10000;          //Transition
parameters which characterize the contour curve
stMoveCircularAbsRadius_in.fTransitionParameter[1]    = 10000;          //Transition
parameters which characterize the contour curve
stMoveCircularAbsRadius_in.ePathChoice                = MC_CLOCKWISE;   //Supported path
choice
stMoveCircularAbsRadius_in.eArcShortLong              = MC_SHORT;//Defines the types of
supported arc length
stMoveCircularAbsRadius_in.eCoordSystem               = MC_MCS_COORD;//Supported coordinate
system
stMoveCircularAbsRadius_in.eTransitionMode            =
MC_TM_DEFINED_VELOCITY_MODE;//Supported transition modes
stMoveCircularAbsRadius_in.eBufferMode                = MC_BUFFERED_MODE;//Defines the
behavior of the axis
stMoveCircularAbsRadius_in.dEndPoint[0]               = 1000.0;//Absolute end point positions
stMoveCircularAbsRadius_in.dEndPoint[1]               = 1000.0;//Absolute end point positions
stMoveCircularAbsRadius_in.fVelocity                  = 5000.0;         //Maximum Velocity of
the path
stMoveCircularAbsRadius_in.ucSuperimposed             = 1;              //Option to
superimpose is operated
stMoveCircularAbsRadius_in.ucExecute                  = 1;
//
rc = MMC_MoveCircularAbsoluteRadiusCmd (hConn, iAxisRef, &stMoveCircularAbsRadius_in,
&stMoveCircularAbs_out);
if (rc != 0)
{
    HandleError() ;
}
```

5.8.12. MMC_MoveCircularAbsoluteAngle

For multi-axis systems. Commands an interpolated circular angle movement on an axes group from the actual position of the TCP.

```
MMC_LIB_API int MMC_MoveCircularAbsoluteAngleCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVECIRCULARABSOLUTEANGLE_IN* pInParam,
OUT MMC_MOVECIRCULARABSOLUTE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MOVECIRCULARABSOLUTEANGLE_IN** group axes input structure that receives the MMC_MoveCircularAbsoluteAngle command.

pOutParam

Points to the **MMC_MOVECIRCULARABSOLUTE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveCircularAbsoluteAngle function.

Remarks

The end-point and auxiliary point, depending on the applied mode, are defined absolutely in the specified coordinate system.

Scope

Motion is only permitted in two dimensions. Only MCS coordinate system is supported.

MOVECIRCULARABSOLUTEANGLE_IN Structure

```
typedef struct{
double dCenterPoint[NC_MAX_NUM_AXES_IN_NODE];
double dAngle;
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucSuperimposed;
unsigned char ucExecute;
}MMC_MOVECIRCULARABSOLUTEANGLE_IN;
```

Parameters

dCenterPoint [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute positions for each dimension in the coordinate system specified by the input signal CoordSystem, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dCenterPoint can have vector array [1….3] double values in a technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

dAngle

Relative angular position for the coordinate system specified by the input signal CoordSystem. Angular double value in degrees [u]

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes**.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes** and options below. The options are:

MC_TM_NONE_MODE = 0

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE = 1

MC_BUFFERED_MODE = 2

MC_BLENDING_LOW_MODE = 3

MC_BLENDING_PREVIOUS_MODE = 4

MC_BLENDING_NEXT_MODE = 5

MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis

will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVECIRCULARABSOLUTE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVECIRCULARABSOLUTE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-90 describes the function block for MMC_MoveCircularAbsoluteAngle

Figure 5-95: MMC_MoveCircularAbsoluteAngle function block

5.8.12.2. Function Block Code Example

```
int rc;
MMC_MOVECIRCULARABSOLUTEANGLE_IN    stMoveCircularAbsAngle_in;
MMC_MOVECIRCULARABSOLUTE_OUT        stMoveCircularAbs_out;
//
// Inserting the structure parameters:
stMoveCircularAbsAngle_in.fAcceleration          = 100000.0;   //Value of the acceleration
stMoveCircularAbsAngle_in.fDeceleration          = 100000.0;   //Value of the deceleration
stMoveCircularAbsAngle_in.fJerk                  = 20000000.0; //Maximum value of the Jerk
stMoveCircularAbsAngle_in.dCenterPoint[0]        = 5000;//Absolute center positions for each
dimension
stMoveCircularAbsAngle_in.dCenterPoint[1]        = 5000;//Absolute center positions for each
dimension
stMoveCircularAbsAngle_in.fTransitionParameter[0] = 10000; //Transition parameters which
characterize the contour curve
stMoveCircularAbsAngle_in.fTransitionParameter[1] = 10000; //Transition parameters which
characterize the contour curve
stMoveCircularAbsAngle_in.dAngle                 = 165;    //absolute positions for each
dimension in the coordinate system
stMoveCircularAbsAngle_in.eCoordSystem           = MC_MCS_COORD;    //Supported coordinate
system
stMoveCircularAbsAngle_in.eTransitionMode        = MC_TM_DEFINED_VELOCITY_MODE;  //Supported
transition modes
stMoveCircularAbsAngle_in.eBufferMode            = MC_BUFFERED_MODE; //Defines the behavior of
the axis
stMoveCircularAbsAngle_in.fVelocity              = 5000.0;          //Maximum Velocity of the
path
stMoveCircularAbsAngle_in.ucSuperimposed         = 1;           //Option to superimpose is
operated
stMoveCircularAbsAngle_in.ucExecute              = 1;
//
rc = MMC_MoveCircularAbsoluteAngleCmd (hConn, iAxisRef, &stMoveCircularAbsAngle_in,
&stMoveCircularAbs_out);
if (rc != 0)
{
    HandleError();
}
```

5.8.13. MMC_MoveLinearAbsolute

For multi-axis systems. Commands an interpolated linear movement on an axes group from the actual position of the TCP to an absolute position in the specified coordinate system.

```
MMC_LIB_API int MMC_MoveLinearAbsoluteCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVELINEARABSOLUTE_IN* pInParam,
OUT MMC_MOVELINEARABSOLUTE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVELINEARABSOLUTE_IN** group axes input structure that receives the MMC_MoveLinearAbsolute command.

pOutParam

Points to the **MMC_MOVELINEARABSOLUTE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveLinearAbsolute function.

Remarks

None

Scope

All

MMC_MOVELINEARABSOLUTE_IN Structure

```
typedef struct{
double dbPosition[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucSuperimposed;
unsigned char ucExecute;
}MMC_MOVELINEARABSOLUTE_IN;
```

Parameters

dbPosition [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute end positions for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbPosition is a vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes**.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1,

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes** and options below. The options are:

MC_TM_NONE_MODE = 0

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE = 1

MC_BUFFERED_MODE = 2

MC_BLENDING_LOW_MODE = 3

MC_BLENDING_PREVIOUS_MODE = 4

MC_BLENDING_NEXT_MODE = 5

MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis

will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVELINEARABSOLUTE_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVELINEARABSOLUTE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-91 describes the function block for MMC_MoveLinearAbsolute

Figure 5-96: MMC_MoveLinearAbsolute function block

5.8.13.2. Function Block Code Example

```
int rc;
MMC_MOVELINEARABSOLUTE_IN       stMoveLinearAbs_in;
MMC_MOVELINEARABSOLUTE_OUT      stMoveLinearAbs_out;
//
// Inserting the structure parameters:
stMoveLinearAbs_in.fAcceleration        = 100000.0;       //Value of the acceleration
stMoveLinearAbs_in.fDeceleration        = 100000.0;       //Value of the deceleration
stMoveLinearAbs_in.fJerk                = 20000000.0;     //Maximum value of the Jerk
stMoveLinearAbs_in.dbPosition[0]        = 5000;           //Absolute position for each
dimension
stMoveLinearAbs_in.dbPosition[1]        = 5000;           //Absolute position for each
dimension
stMoveLinearAbs_in.fTransitionParameter[0]= 10000;            //Transition parameters which
characterize the contour curve
stMoveLinearAbs_in.fTransitionParameter[1]= 10000;            //Transition parameters which
characterize the contour curve
stMoveLinearAbs_in.eCoordSystem         = MC_MCS_COORD;   //Supported coordinate system
stMoveLinearAbs_in.eTransitionMode      = MC_TM_DEFINED_VELOCITY_MODE;//Supported transition
modes
stMoveLinearAbs_in.eBufferMode          = MC_BUFFERED_MODE;//Defines the behavior of the axis
stMoveLinearAbs_in.fVelocity            = 5000.0;         //Maximum Velocity of the path
stMoveLinearAbs_in.ucSuperimposed       = 1;              //Option to superimpose is operated
stMoveLinearAbs_in.ucExecute            = 1;
//
rc = MMC_MoveLinearAbsoluteCmd (hConn, iAxisRef, &stMoveLinearAbs_in, &stMoveLinearAbs_out);
if (rc != 0)
{
    HandleError();
}
```

5.8.13.3. Implementation Example

Figure 5-97: MMC_MoveLinearAbsolute - Example

Figure 5-98: MMC_MoveLinearAbsolute timing diagram - Example

Figure 5-99: Sequence of two complete motions (Done>Execute)

Figure 5-100: Second motion interrupts first motion

5.8.14. MMC_MoveLinearRelative

For multi-axis systems. Commands an interpolated linear movement on an axes group from the actual position of the TCP to a relative distance in the specified coordinate system.

```
MMC_LIB_API int MMC_MoveLinearRelativeCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVELINEARRELATIVE_IN* pInParam,
OUT MMC_MOVELINEARRELATIVE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVELINEARRELATIVE_IN** group axes input structure that receives the MMC_MoveLinearRelative command.

pOutParam

Points to the **MMC_MOVELINEARRELATIVE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveLinearRelative function.

Remarks

None

Scope

All

MMC_MOVELINEARRELATIVE_IN Structure

```
typedef struct{
double dbDistance[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucSuperimposed;
unsigned char ucExecute;
}MMC_MOVELINEARRELATIVE_IN;
```

Parameters

dbDistance [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of relative distances for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbDistance is a vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes.**

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes**. The options are:

MC_TM_NONE_MODE = 0

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE = 1

MC_BUFFERED_MODE = 2

MC_BLENDING_LOW_MODE = 3

MC_BLENDING_PREVIOUS_MODE = 4

MC_BLENDING_NEXT_MODE = 5

MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis

will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVELINEARRELATIVE_OUT Structure

```
typedef struct mmc_movealinearrelative_out{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVELINEARRELATIVE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-96 describes the function block for MMC_MoveLinearRelative.

Figure 5-101: MMC_MoveLinearRelative function block

5.8.14.2. Function Block Code Example

```
int rc;
MMC_MOVELINEARRELATIVE_IN       stMoveLinearRel_in;
MMC_MOVELINEARRELATIVE_OUT      stMoveLinearRel_out;
//
// Inserting the structure parameters:
stMoveLinearRel_in.fAcceleration              = 100000.0;            //Value of the acceleration
stMoveLinearRel_in.fDeceleration              = 100000.0;            //Value of the deceleration
stMoveLinearRel_in.fJerk                      = 20000000.0;          //Maximum value of the Jerk
stMoveLinearRel_in.dbDistance[0]              = 10000;      //Absolute position for each
dimension
stMoveLinearRel_in.dbDistance[1]              = 10000;      //Absolute position for each
dimension
stMoveLinearRel_in.fTransitionParameter[0]    = 10000;               //Transition parameters
which characterize the contour curve
MoveLinearRel_in.fTransitionParameter[1]      = 10000;               //Transition parameters
which characterize the contour curve
stMoveLinearRel_in.eCoordSystem               = MC_MCS_COORD;        //Supported coordinate
system
stMoveLinearRel_in.eTransitionMode            = MC_TM_DEFINED_VELOCITY_MODE;  //Supported
transition modes
stMoveLinearRel_in.eBufferMode                = MC_BUFFERED_MODE;    //Defines the behavior of
the axis
stMoveLinearRel_in.fVelocity                  = 5000.0;              //Maximum Velocity of the
path
stMoveLinearRel_in.ucSuperimposed             = 1;              //Option to superimpose is
operated
stMoveLinearRel_in.ucExecute                  = 1;
//
rc = MMC_MoveLinearRelativeCmd (hConn, iAxisRef, &stMoveLinearRel_in, &stMoveLinearRel_out);
if (rc != 0)
{
    HandleError();
}
```

5.8.14.3. Implementation Example

Figure 5-102: MMC_MoveLinearRelative - Example

Figure 5-103: Timing diagram for MMC_MoveLinearRelative – Example

Timing diagram for example above (the dots on the red line are based on the same timing difference and representing the velocity)

Figure 5-104: Sequence of two complete motions

Figure 5-105: Sequence where second motion interrupts first motion

5.8.15. MMC_MoveLinearAdditive

For multi-axis systems. Commands an interpolated linear movement on an axes group from the actual position of the TCP to an additive position in the specified coordinate system.

```
MMC_LIB_API int MMC_MoveLinearAdditiveCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVELINEARADDITIVE_IN* pInParam,
OUT MMC_MOVELINEARADDITIVE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVELINEARADDITIVE_IN** group axes input structure that receives the MMC_MoveLinearAdditive command.

pOutParam

Points to the **MMC_MOVELINEARADDITIVE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveLinearAdditive function.

Remarks

None

Scope

All

MMC_MOVELINEARADDITIVE_IN Structure

```
typedef struct{
double dbDistance[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity
float fAcceleration
float fDeceleration
float fJerk
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE]
MC_COORD_SYSTEM_ENUM eCoordSystem
NC_TRANSITION_MODE_ENUM eTransitionMode
MC_BUFFERED_MODE_ENUM eBufferMode
unsigned char ucSuperimposed
unsigned char ucExecute
}MMC_MOVELINEARADDITIVE_IN;
```

Parameters

dbDistance [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of relative distances for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbDistance is a vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units accuracy to 4 bytes only, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes**.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes** and options below. The options are:

MC_TM_NONE_MODE = 0

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE = 1

MC_BUFFERED_MODE = 2

MC_BLENDING_LOW_MODE = 3

MC_BLENDING_PREVIOUS_MODE = 4

MC_BLENDING_NEXT_MODE = 5

MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVELINEARADDITIVE_OUT Structure

```
typedef struct mmc_movealinearadditive_out{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVELINEARADDITIVE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-96 describes the function block for MMC_MoveLinearAdditive.

Figure 5-106: MMC_MoveLinearAdditive function block

5.8.15.2. Function Block Code Example

5.8.16. MMC_MoveLinearAdditiveEx

For multi-axis systems. Commands an extended interpolated linear movement on an axes group from the actual position of the TCP to an additive accurate position in the specified coordinate system.

```
MMC_LIB_API int MMC_MoveLinearAdditiveExCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVELINEARADDITIVEEX_IN* pInParam,
OUT MMC_MOVELINEARADDITIVEEX_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVELINEARADDITIVEEX_IN** group axes input structure that receives the MMC_MoveLinearAdditiveEx command.

pOutParam

Points to the **MMC_MOVELINEARADDITIVEEX_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveLinearAdditiveEx function.

Remarks

None

Scope

All

MMC_MOVELINEARADDITIVEEX_IN Structure

```
typedef struct{
double dbDistance[NC_MAX_NUM_AXES_IN_NODE
double dVelocity
double dAcceleration;
double dDeceleration;
double dJerk
double dTransitionParameter[NC_MAX_NUM_AXES_IN_NODE
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode
unsigned char ucSuperimposed
unsigned char ucExecute;
}MMC_MOVELINEARADDITIVEEX_IN;
```

Parameters

dbDistance [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of relative distances for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbDistance is a vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

dVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive double value in u/s

dAcceleration

Value of the acceleration (increasing energy of the motor). Any positive double value in $u/s^{2.}$

dDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive double value in u/s^2

dJerk

Maximum double value of the Jerk. Any positive value in u/s^3

dTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dTransitionParameter can have any positive double value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **These sections describe the special functions and their parameters which perform a transition or circular motion of a single or group axes.**

Transition Modes.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section below **5.8.2.1 Transition Modes**. The options are:

MC_TM_NONE_MODE = 0

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE = 1

MC_BUFFERED_MODE = 2

MC_BLENDING_LOW_MODE = 3

MC_BLENDING_PREVIOUS_MODE = 4

MC_BLENDING_NEXT_MODE = 5

MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous

movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVELINEARADDITIVEEX_OUT Structure

```
typedef struct mmc_movealinearadditiveex_out{
unsigned int uiHndl
unsigned short usStatus;
short usErrorID;
}MMC_MOVELINEARADDITIVEEX_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-96 describes the function block for MMC_MoveLinearAdditiveEx.

Figure 5-107: MMC_MoveLinearAdditiveEx function block

5.8.16.2. Function Block Code Example

5.8.17. MMC_MoveLinearAbsoluteRepetitive

For multi-axis systems. Commands an interpolated repetitive linear movement on an axes group vector to an absolute point given as an input in the specified coordinate system.

```
MMC_LIB_API int MMC_MoveLinearAbsoluteRepetitiveCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVELINEARABSOLUTEREPETITIVE_IN* pInParam,
OUT MMC_MOVELINEARABSOLUTEREPETITIVE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVELINEARABSOLUTEREPETITIVE_IN** group axes input structure that receives the MMC_MoveLinearAbsoluteRepetitive command.

pOutParam

Points to the **MMC_MOVELINEARRELATIVEREPETITIVE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveLinearAbsoluteRepetitive function.

Remarks

It should be noted that all repetitive mode motion will not operate in Abort Mode.

Scope

This function only operates when the vector state is in Standby mode. Calling MMC_Stop, MMC_Halt in aborting mode will stop the motion.

MMC_MOVELINEARABSOLUTEREPETITIVE_IN Structure

```
typedef struct mmc_movelinearabsoluterepetitive_in{
double dbPosition[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned int uiExecDelayMs;
unsigned char ucSuperImposed;
unsigned char ucExecute;
}MMC_MOVELINEARABSOLUTEREPETITIVE_IN;
```

Parameters

dbPosition [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute end positions for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbPosition is a vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section ***5.8.2.1 Transition Modes.***

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes**. The options are:

MC_TM_NONE_MODE = 0

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

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

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE = 1

MC_BUFFERED_MODE = 2

MC_BLENDING_LOW_MODE = 3

MC_BLENDING_PREVIOUS_MODE = 4

MC_BLENDING_NEXT_MODE = 5

MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous

function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

uiExecDelayMs

The delay in execution of the next action (in msecs). Any +ve integer value.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVELINEARABSOLUTEREPETITIVE_OUT Structure

```
typedef struct mmc_movelinearabsoluterepetitive_out{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVELINEARABSOLUTEREPETITIVE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-103 describes the function block for MMC_MoveLinearAbsoluteRepetitive

Figure 5-108: MMC_MoveLinearAbsoluteRepetitive function block

5.8.17.2. Function Block Code Example

```
int rc;
MMC_MOVELINEARABSOLUTEREPETITIVE_IN      stMoveLinAbsRepIn;
MMC_MOVELINEARABSOLUTEREPETITIVE_OUT     stMoveLinAbsRepOut;
// Suppose we are working in 3D
stMoveLinAbsRepIn.dbPosition[0]              = 100000.0;
stMoveLinAbsRepIn.dbPosition[1]              = 100000.0;
stMoveLinAbsRepIn.dbPosition[2]              = 100000.0;
stMoveLinAbsRepIn.eBufferMode                = MC_BUFFERED_MODE_ENUM;
stMoveLinAbsRepIn.eTransitionSystem          = MC_TM_NONE_MODE;
stMoveLinAbsRepIn.fTransitionParameter[0]    = 0.0;
stMoveLinAbsRepIn.fAcceleration              = 1000000.0;
stMoveLinAbsRepIn.fDeceleration              = 1000000.0;
stMoveLinAbsRepIn.fJerk                      = 10000000.0;
stMoveLinAbsRepIn.fVelocity                  = 100000.0;
stMoveLinAbsRepIn.ucExecute                  = 1;
stMoveLinAbsRepIn.uiExecDelayMs              = 0;
//
rc = MMC_MoveLinearAbsoluteRepetitiveCmd (hConn, iAxisRef, &stMoveLinAbsRepIn,
&stMoveLinAbsRepOut);
if (rc != 0)
{
HandleError();
}
```

5.8.18. MMC_MoveLinearRelativeRepetitive

For multi-axis systems. Commands an interpolated repetitive linear movement on an axes group vector to a relative distance from actual position given as an input in the specified coordinate system.

```
MMC_LIB_API int MMC_MoveLinearRelativeRepetitiveCmd
(IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVELINEARRELATIVEREPETITIVE_IN* pInParam,
OUT MMC_MOVELINEARRELATIVEREPETITIVE_OUT* pOutParam
);
```

Motion Mode NC - Supported Distributed - Not Supported

Source GMAS\includes\MMC_PLCopen_group_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVELINEARRELATIVEREPETITIVE_IN** group axes input structure that receives the MMC_MoveLinearRelativeRepetitive command.

pOutParam

Points to the **MMC_MOVELINEARRELATIVEREPETITIVE_OUT** group axes output structure receiving information, as a result of calling the MMC_MoveLinearRelativeRepetitive function.

Remarks

It should be noted that all repetitive mode motion will not operate in Abort Mode.

Scope

This function only operates when the vector state is in Standby mode. Calling MMC_Stop, MMC_Halt in aborting mode will stop the motion.

MMC_MOVELINEARRELATIVEREPETITIVE_IN Structure

```
typedef struct mmc_movelinearrelativerepetitive_in{
double dbDistance[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_COORD_SYSTEM_ENUM eCoordSystem;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned int uiExecDelayMs;
unsigned char ucSuperImposed;
unsigned char ucExecute;
}MMC_MOVELINEARRELATIVEREPETITIVE_IN;
```

Parameters

dbDistance[NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of relative distances for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbDistance is a vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

fVelocity

Value of the maximum velocity (not necessarily reached) in which the path is defined. Any positive float value in u/s

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE = 1

MC_BUFFERED_MODE = 2

MC_BLENDING_LOW_MODE = 3

MC_BLENDING_PREVIOUS_MODE = 4

MC_BLENDING_NEXT_MODE = 5

MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

uiExecDelayMs

The delay in execution of the next action (in msecs). Any +ve integer value.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVELINEARRELATIVEREPETITIVE_OUT Structure

```
typedef struct mmc_movelinearrelativerepetitive_out{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVELINEARRELATIVEREPETITIVE_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Displays an error code as -ve or +ve integers.

Figure 5-104 describes the function block for MMC_MoveLinearRelativeRepetitive

Figure 5-109: MMC_MoveLinearRelativeRepetitive function block

5.8.18.2. Function Block Code Example

```
int rc;
MMC_MOVELINEARRELATIVEREPETITIVE_IN      stMoveLinRelRepIn;
MMC_MOVELINEARRELATIVEREPETITIVE_OUT     stMoveLinAbsRepOut;
// Suppose we are working in 3D
stMoveLinRelRepIn.dbDistance[0]          = 100000.0;
stMoveLinRelRepIn.dbDistance[1]          = 100000.0;
stMoveLinRelRepIn.dbDistance[2]          = 100000.0;
stMoveLinRelRepIn.eBufferMode            = MC_BUFFERED_MODE_ENUM;
stMoveLinRelRepIn.eTransitionSystem      = MC_TM_NONE_MODE;
stMoveLinRelRepIn.fTransitionParameter[0]= 0.0;
stMoveLinRelRepIn.fAcceleration          = 1000000.0;
stMoveLinRelRepIn.fDeceleration          = 1000000.0;
stMoveLinRelRepIn.fJerk                  = 10000000.0;
stMoveLinRelRepIn.fVelocity              = 100000.0;
stMoveLinRelRepIn.ucExecute              = 1;
stMoveLinRelRepIn.uiExecDelayMs          = 0;
//
rc = MMC_MoveLinearRelativeRepetitiveCmd (hConn, iAxisRef, &stMoveLinRelRepIn,
&stMoveLinRelRepOut);
if (rc != 0)
{
HandleError();
}
```

5.8.19. MMC_PathSelect

For multi-axis systems. Reads splines data from a file and calculates the optimal path.

```
MMC_LIB_API int MMC_PathSelectCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_PATHSELECT_IN* pInParam,
OUT MMC_PATHSELECT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Group reference handle type returned by GetAxisRef command in the vector motion.

pInParam

Points to the **MMC_PATHSELECT_IN** input data structure using the MMC_PathSelect function.

pOutParam

Points to the **MMC_PATHSELECT_OUT** output structure receiving information, as a result of calling the MMC_PathSelect function.

Remarks

This function obtains the spline file from the user, calculates the trajectory and stores it in a specially reserved memory. In other words, this function prepares the trajectory for MMC_MovePathCmd().

Scope

All

MMC_PATHSELECT_IN Structure

```
typedef struct mmc_pathselect_in{
MC_COORD_SYSTEM_ENUM eCoordSystem;
MC_PATH_DATA_REF pPathToSplineFile;
unsigned char ucExecute;
}MMC_PATHSELECT_IN;
```

Parameters

eCoordSystem

Define the types of supported coordinate systems using the MC_COORD_SYSTEM_ENUM enumerator. The options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

pPathToSplineFile

This string describes where the splines data file is located, the absolute path. Values accepted are any characters describing a file path, for example: /mnt/jffs/usr/Spline_2D.txt

MC_PATH_DATA_REF Where MC_PATH_DATA_REF is a string of NC_MAX_SPLINES_FILE_PATH_LENGTH that defines the maximum length of the splines file path data.

MC_PATH_DATA_REF is an alias for unsigned integer and can have values that are allowed by Linux for a file name.

NC_MAX_SPLINES_FILE_PATH_LENGTH is a constant of value 100.

ucExecute

Start the execution command. Boolean TRUE/FALSE values. Currently not in use, and set to 1.

MMC_PATHSELECT_OUT Structure

```
typedef struct mmc_pathselect_out{
unsigned short usStatus;
short usErrorID;
MC_PATH_REF hMemHandle;
}MMC_PATHSELECT_OUT;
```

Parameters

hMemHandle

MC_PATH_REF alias for unsigned integer with values and handle to a journal entry where the pointer to the shared memory is located, and indicates the memory area where the spline is allocated. In fact this is the unique identifier between PathSelect, MovePath and PathUnselect commands. MC_PATH_REF is the journal entry path reference.

hMemHandle produces +ve Integer values.

usStatus

Bitwise returned command status with the values MMC_REMOTE_FUNC_STATUS_BIT_ERROR or 0 (OK).

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-105 describes the function block for MMC_PathSelect.

Figure 5-110: MMC_PathSelect function block

5.8.19.2. Function Block Code Example

```
int rc;
MMC_PATHSELECT_IN      stPathSelectIn;
MMC_PATHSELECT_OUT     stPathSelectOut;
// Inserting the structure parameters:
stPathSelectIn.eCoordSystem = MC_MCS_COORD;
stPathSelectOut.usErrorID = 0;
strcpy(stPathSelectIn.pPathToSplineFile, "/mnt/jffs/usr/Helix.txt");
stPathSelectIn.ucExecute = 1;
retval = 0;
retval = MMC_PathSelectCmd(g_conn_hndl, g_vect_ref[vect_idx], &stPathSelectIn,
&stPathSelectOut);
if (0 > retval)
{
printf("Error ID: %d", (short )stPathSelectOut.usErrorID);
return -1;
}
```

5.8.20. MMC_MovePath

For multi-axis systems. Moves a group of drives along a previously defined spline path.

```
MMC_LIB_API int MMC_MovePathCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVEPATH_IN* pInParam,
OUT MMC_MOVEPATH_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Group reference handle type returned by GetAxisRef command in the vector motion.

pInParam

Points to the **MMC_MOVEPATH_IN** input data structure using the MMC_MovePath function.

pOutParam

Points to the **MMC_MOVEPATH_OUT** output structure receiving information, as a result of calling the MMC_MovePath function.

Remarks

Note that only buffered mode, motion from STANDBY state, with no transitions, is currently supported.

Scope

All

MMC_MOVEPATH_IN Structure

```
typedef struct mmc_movepath_in{
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
MC_PATH_REF hMemHandle;
unsigned char ucSuperImposed;
unsigned char ucExecute;
} MMC_MOVEPATH_IN;
```

Parameters

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes**.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems using the MC_COORD_SYSTEM_ENUM enumerator. The options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes**. The options are:

MC_TM_NONE_MODE = 0, only this mode supported at this time

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

MC_TM_DEFINED_VELOCITY_MODE = 2, Not supported at this time

MC_TM_CORNER_DISTANCE_MODE = 3, Not supported at this time

MC_TM_MAX_CORNER_DEVIATION_MODE = 4, Not supported at this time

MC_TM_SWITCH_RADIUS_MODE = 5, Not supported at this time

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Buffered Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2, only this mode supported
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

hMemHandle

MC_PATH_REF index handle to a journal entry where the pointer to the shared memory is located obtained from the PathSelect command. MC_PATH_REF is the journal entry path reference.

hMemHandle has Integer values.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE. Not currently in use.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVEPATH_OUT Structure

```
typedef struct mmc_movepath_out{
unsigned int uiHandle;
unsigned short usStatus;
short usErrorID;
}MMC_MOVEPATH_OUT;
```

Parameters

uiHandle

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the values MMC_REMOTE_FUNC_STATUS_BIT_ERROR or 0 (OK).

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-106 describes the function block for MMC_MovePath

Figure 5-111: MMC_MovePath function block

5.8.20.2. Function Block Code Example

```
int rc;
MMC_MOVEPATH_IN   stMovePathIn;
MMC_MOVEPATH_OUT  stMovePathOut;
// Inserting the structure parameters:
stMovePathIn.fTransitionParameter[0] = 0.0;
stMovePathIn.eCoordSystem = MC_MCS_COORD;
stMovePathIn.eTransitionMode = MC_TM_NONE_MODE;
stMovePathIn.eBufferMode = MC_BUFFERED_MODE;
stMovePathIn.hMemHandle = stPathSelectOut.hMemHandle;
stMovePathIn.ucSuperImposed = 0;
stMovePathIn.ucExecute = 1;
retval = MMC_MovePathCmd(g_conn_hndl, g_vect_ref[vect_idx], &stMovePathIn, &stMovePathOut);
if (0 > retval)
{
printf("Error ID: %d", (short )stMovePathOut.usErrorID);
return -1;
}
```

5.8.20.3. Implementation Example

```
MMC_PATHSELECT_IN stPathSelectIn;
MMC_PATHSELECT_OUT stPathSelectOut;

MMC_MOVEPATH_IN stMovePathIn;
MMC_MOVEPATH_OUT stMovePathOut;

MMC_PATHUNSELECT_IN stPathUnselectIn;
MMC_PATHUNSELECT_OUT stPathUnselectOut;

int rc = 0;
stPathSelectIn.eCoordSystem = MC_MCS_COORD;
strcpy(stPathSelectIn.pPathToSplineFile, "/mnt/jffs/usr/Helix_CL_CV.txt");
stPathSelectIn.ucExecute = 1;

rc = MMC_PathSelectCmd(g_hConnectHandle, g_hVectorRef, &stPathSelectIn, &stPathSelectOut);
if (NC_OK != rc)
{
      HandleError();
      return -1;
}
stMovePathIn.eBufferMode = MC_BUFFERED_MODE;
stMovePathIn.eCoordSystem = MC_MCS_COORD;
stMovePathIn.eTransitionMode = MC_TM_NONE_MODE;
stMovePathIn.fTransitionParameter[0] = 0;
stMovePathIn.hMemHandle = stPathSelectOut.hMemHandle;
stMovePathIn.ucExecute = 1;
stMovePathIn.ucSuperImposed = 0;

rc = MMC_MovePathCmd(g_hConnectHandle, g_hVectorRef, &stMovePathIn, &stMovePathOut);
if (NC_OK != rc)
{
HandleError();
      return -1;
}
stPathUnselectIn.hMemHandle = stPathSelectOut.hMemHandle;
stPathUnselectIn.ucExecute = 1;

rc = MMC_PathUnselectCmd(g_hConnectHandle, g_hVectorRef, &stPathUnselectIn,
&stPathUnselectOut);
if (NC_OK != rc)
{
HandleError();
      return -1;
}
```

5.8.21. MMC_PathUnselect

For multi-axis systems. This function unloads the spline data table from the G-MAS.

```
MMC_LIB_API int MMC_PathUnselectCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_PATHUNSELECT_IN* pInParam,
OUT MMC_PATHUNSELECT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Group reference handle type returned by GetAxisRef command in the vector motion.

pInParam

Points to the **MMC_PATHUNSELECT_IN** input data structure using the MMC_PathUnselect function.

pOutParam

Points to the **MMC_PATHUNSELECT_OUT** output structure receiving information, as a result of calling the MMC_PathUnselect function.

Remarks

None

Scope

All

MMC_PATHUNSELECT_IN Structure

```
typedef struct mmc_pathunselect_in{
MC_PATH_REF hMemHandle;
unsigned char ucExecute;
}MMC_PATHUNSELECT_IN
```

Parameters

hMemHandle

MC_PATH_REF enumerator handle to a journal entry where the pointer to the shared memory is located, obtained from the PathSelect command. MC_PATH_REF is the journal entry path reference.

hMemHandle can have Integer values.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values. Currently not in use and set to 1.

MMC_PATHUNSELECT_OUT Structure

```
typedef struct mmc_pathunselect_out{
unsigned short usStatus;
short usErrorID;
}MMC_PATHUNSELECT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the values MMC_REMOTE_FUNC_STATUS_BIT_ERROR or 0 (OK).

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-107 describes the function block for MMC_PathUnselect.

Figure 5-112: MMC_PathUnselect function block

5.8.21.2. Function Block Code Example

```
int rc;
MMC_PATHUNSELECT_IN    stPathUnselectIn;
MMC_PATHUNSELECT_OUT   stPathUnselectOut;
// Inserting the structure parameters:
stPathUnselectIn.hMemHandle = stPathSelectOut.hMemHandle;
stPathUnselectIn.ucExecute = 1;
retval = MMC_PathUnselectCmd(g_conn_hndl, g_vect_ref[vect_idx], &stPathUnselectIn,
&stPathUnselectOut);
if (0 > retval)
{
printf("Error ID: %d", (short )stPathUnselectOut.usErrorID);
return -1;
}
```

5.9. Multiple Axes Administrative Control

Administrative function blocks are blocks where no driving motion is involved. The following multiple axes administrative function blocks are described:

Multiple Axes
MMC_AddAxisToGroup
MMC_GroupDisable
MMC_GroupEnable
MMC_GroupReadActualPosition
MMC_GroupReadActualVelocity
MMC_GroupReadError
MMC_GroupReadStatus
MMC_GroupReset
MMC_GroupSetOverride
MMC_GroupSetPosition
MMC_RemoveAxisFromGroup
MMC_SetKinTransform
MMC_GroupReadParameter
MMC_GroupReadBoolParameter
MMC_GroupWriteParameter
MMC_GroupWriteBoolParameter

5.9.1. MMC_AddAxisToGroup

For multiple Axes. Adds one axis to a group in a structure *AxesGroup*.

```
MMC_LIB_API int MMC_AddAxisToGroup(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_ADDAXISTOGROUP_IN* pInParam,
OUT MMC_ADDAXISTOGROUP_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_ADDAXISTOGROUP_IN** input data structure using the MMC_AddAxisToGroup function.

pOutParam

Points to the **MMC_ADDAXISTOGROUP_OUT** output structure receiving information, as a result of calling the MMC_AddAxisToGroup function.

Remarks

This is an administrative function block, since no movement is generated. The command cannot be buffered.

Each *IdentInGroup* can be used only once, otherwise it leads to an error.

Scope

An axis can be added to a group in GroupStandby or GroupDisabled, but not in GroupHoming, GroupStopping, or GroupErrorStop.

If an axis group is previously defined with nodes, in the resource file, do not call this function. Calling this function under this circumstance will result in the error -69.

MMC_ADDAXISTOGROUP_IN Structure

```
typedef struct{
NC_NODE_HNDL_T hNode;
NC_IDENT_IN_GROUP_ENUM eIdentInGroup;
}MMC_ADDAXISTOGROUP_IN;
```

Parameters

hNode

The NC_NODE_HNDL_T enumerator defines the Node handle transition. The axis ref parameter

hNode can have any positive numeric value.

eIdentInGroup

The NC_IDENT_IN_GROUP_ENUM enumerator identifies the order and Nodes in the group of the added axis. Performed via an enumerator to give the different axes a name in the order, which can be coupled to the names in the kinematic model. The options are:

NC_NODE_1_ID = 0

……

NC_NODE_16_ID = 15

MMC_ADDAXISTOGROUP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_ADDAXISTOGROUP_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-108 describes the function block for MMC_AddAxisToGroup

Figure 5-113: MMC_AddAxisToGroup function block

5.9.1.2. Function Block Code Example

```
int rc;
MMC_ADDAXISTOGROUP_IN      stAddAxisToGroup_in;
MMC_ADDAXISTOGROUP_OUT     stAddAxisToGroup_out;
//
// Inserting the structure parameters:
stAddAxisToGroup_in.hNode           = 2;              //Defines the Node handle transition
stAddAxisToGroup_in.eIdentInGroup   = NC_NODE_4_ID;   //Identifies the order and Nodes in the
group of the added axis
//
rc = MMC_AddAxisToGroup (hConn, iAxisRef, &stAddAxisToGroup_in, &stAddAxisToGroup_out);
if (rc != 0)
{
    HandleError();
}
```

5.9.2. MMC_GroupDisable

For multiple Axes. Changes the state for a group to GroupDisabled, although it is an administrative function block, since no movement is generated.

```
MMC_LIB_API int MMC_GroupDisableCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPDISABLE_IN* pInParam,
OUT MMC_GROUPDISABLE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPDISABLE_IN** input data structure using the MMC_GroupDisable function.

pOutParam

Points to the **MMC_GROUPDISABLE_OUT** output structure receiving information, as a result of calling the MMC_GroupDisable function.

Remarks

If the axes are not static while issuing this command, the application should take the necessary precautions.

Scope

All

MMC_GROUPDISABLE_IN Structure

```
typedef struct{
unsigned char ucExecute;
}MMC_GROUPDISABLE_IN;
```

Parameters

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_GROUPDISABLE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_GROUPDISABLE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-109 describes the function block for MMC_GroupDisable.

Figure 5-114: MMC_GroupDisable function block

5.9.2.2. Function Block Code Example

```
int rc;
MMC_GROUPDISABLE_IN        stGroupDisable_in;
MMC_GROUPDISABLE_OUT       stGroupDisable_out;
//
// Inserting the structure parameters:
stGroupDisable_in.ucExecute     = 0;     // Start the execution command

//
rc = MMC_GroupDisableCmd (hConn, iAxisRef, &stGroupDisable_in, &stGroupDisable_out);
if (rc != 0)
{
    HandleError();
}
```

5.9.3. MMC_GroupEnable

For multi-axis systems. Changes the state for a group from GroupDisabled to GroupStandby. This is an administrative function block, since no movement is generated.

```
MMC_LIB_API int MMC_GroupEnableCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPENABLE_IN* pInParam,
OUT MMC_GROUPENABLE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPENABLE_IN** input data structure using the MMC_GroupEnable function.

pOutParam

Points to the **MMC_GROUPENABLE_OUT** output structure receiving information, as a result of calling the MMC_GroupEnable function.

Remarks

The command does not influence the power state of any of the single axes in the group.

Scope

All

MMC_GROUPENABLE_IN Structure

```
typedef struct{
unsigned char ucExecute;
}MMC_GROUPENABLE_IN;
```

Parameters

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_GROUPENABLE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_GROUPENABLE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-110 describes the function block for MMC_GroupEnable.

Figure 5-115: MMC_GroupEnable function block

5.9.3.2. Function Block Code Example

```
int rc;
MMC_GROUPENABLE_IN          stGroupEnable_in;
MMC_GROUPENABLE_OUT         stGroupEnable_out;
//
// Inserting the structure parameters:
stGroupEnable_in.ucExecute      = 1;     // Start the execution command

//
rc = MMC_GroupEnableCmd (hConn, iAxisRef, &stGroupEnable_in, &stGroupEnable_out);
if (rc != 0)
{
    HandleError();
}
```

5.9.4. MMC_GroupReadActualPosition

For multi-axis systems. Returns the actual position in the selected coordinate system of an axes group. This is an administrative function block, since no movement is generated.

```
MMC_LIB_API int MMC_GroupReadActualPosition(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPREADACTUALPOSITION_IN* pInParam,
OUT MMC_GROUPREADACTUALPOSITION_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPREADACTUALPOSITION_IN** input data structure using the MMC_GroupReadActualPosition function.

pOutParam

Points to the **MMC_GROUPREADACTUALPOSITION_OUT** output structure receiving information, as a result of calling the MMC_GroupReadActualPosition function.

Remarks

None

Scope

All

MMC_GROUPREADACTUALPOSITION_IN Structure

```
typedef struct{
MC_COORD_SYSTEM_ENUM eCoordSystem;
unsigned char ucEnable;
}MMC_GROUPREADACTUALPOSITION_IN;
```

Parameters

eCoordSystem

Define the types of supported coordinate systems using the MC_COORD_SYSTEM_ENUM enumerator. The options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_GROUPREADACTUALPOSITION_OUT Structure

```
typedef struct{
double dbPosition[NC_MAX_NUM_AXES_IN_NODE];
unsigned short usStatus;
short usErrorID;
}MMC_GROUPREADACTUALPOSITION_OUT;
```

Parameters

dbPosition [NC_MAX_NUM_AXES_IN_NODE]

Array [1..N] of absolute end positions for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbPosition is a vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-111 describes the function block for MMC_GroupReadActualPosition

Figure 5-116: MMC_GroupReadActualPosition function block

5.9.4.2. Function Block Code Example

```
int rc;
MMC_GROUPREADACTUALPOSITION_IN      stGroupReadActualPos_in;
MMC_GROUPREADACTUALPOSITION_OUT     stGroupReadActualPos_out;
//
// Inserting the structure parameters:
stGroupReadActualPos_in.eCoordSystem     = MC_MCS_COORD; //Define types of supported coordinate
systems
stGroupReadActualPos_in.ucEnable         = 1;            //Get the actual position in the
selected coordinate system of the axes group
//
rc = MMC_GroupReadActualPosition (hConn, iAxisRef, &stGroupReadActualPos_in,
&stGroupReadActualPos_out);
printf("Group Actual Position[%ld] ErrId[%d]\n", (long
int)stGroupReadActualPos_out.dbPosition, (short)stGroupReadActualPos_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.9.5. MMC_GroupReadActualVelocity

Returns the actual velocity in the selected coordinate system of an axes group. This is an administrative function block, since no movement is generated.

```
MMC_LIB_API int MMC_GroupReadActualVelocity(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPREADACTUALVELOCITY_IN* pInParam,
OUT MMC_GROUPREADACTUALVELOCITY_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPREADACTUALVELOCITY_IN** input data structure using the MMC_GroupReadActualVelocity function.

pOutParam

Points to the **MMC_GROUPREADACTUALVELOCITY_OUT** output structure receiving information, as a result of calling the MMC_GroupReadActualVelocity function.

Remarks

None

Scope

Relevant for ACS coordinate system only and not MCS. Conditional on the values of the CANbus PDO3 and PDO4.

MMC_GROUPREADACTUALVELOCITY_IN Structure

```
typedef struct{
MC_COORD_SYSTEM_ENUM eCoordSystem;
unsigned char ucEnable;
}MMC_GROUPREADACTUALVELOCITY_IN;
```

Parameters

eCoordSystem

Define the types of supported coordinate systems using the MC_COORD_SYSTEM_ENUM enumerator. The options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_GROUPREADACTUALVELOCITY_OUT Structure

```
typedef struct{
double dVelocity[NC_MAX_NUM_AXES_IN_NODE];
double dPathVelocity;
unsigned short usStatus;
short usErrorID;
}MMC_GROUPREADACTUALVELOCITY_OUT;
```

Parameters

dVelocity [NC_MAX_NUM_AXES_IN_NODE]

Current velocity of the group:

- For ACS the velocities of the different axes
- For MCS it provides the velocity of the TCP

The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dVelocity Any -ve or +ve array double value in the axis's unit [u/s]

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

dPathVelocity

Current path velocity (speed, combined result) of the TCP. *dPathVelocity* can haver values of any -ve or +ve double values in axis's unit [u].

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-112 describes the function block for MMC_GroupReadActualVelocity

Figure 5-117: MMC_GroupReadActualVelocity function block

5.9.5.2. Function Block Code Example

```
int rc;
MMC_GROUPREADACTUALVELOCITY_IN      stGroupReadActualVel_in;
MMC_GROUPREADACTUALVELOCITY_OUT     stGroupReadActualVel_out;
//
// Inserting the structure parameters:
stGroupReadActualVel_in.eCoordSystem     = MC_MCS_COORD; //Define types of supported coordinate
systems
stGroupReadActualVel_in.ucEnable         = 1;            //Get the actual position in the
selected coordinate system of the axes group
//
rc = MMC_GroupReadActualVelocity (hConn, iAxisRef, &stGroupReadActualVel_in,
&stGroupReadActualVel_out);
printf("Group Actual Velocity[%ld] ErrId[%d]\n", (long int)stGroupReadActualVel_out.dVelocity,
(short)stGroupReadActualVel_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.9.6. MMC_GroupReadError

Describes general axes group errors not relating to the function blocks. This is an administrative function block, since no movement is generated. This function is not in use at this moment.

```
MMC_LIB_API int MMC_GroupReadError(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPREADERROR_IN* pInParam,
OUT MMC_GROUPREADERROR_OUT* pOutParam
);
```

Motion Mode NC - Supported Distributed - Not Supported

Source GMAS\includes\MMC_PLCopen_group_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPREADERROR_IN** input data structure using the MMC_GroupReadError function.

pOutParam

Points to the **MMC_GROUPREADERROR_OUT** output structure receiving information, as a result of calling the MMC_GroupReadError function.

Remarks

Examples are (software) limit switch exceeded or single axis error in GroupStandby state. This function is not in use at this moment and will always produce a usGroupErrorID of 256.

Scope

MMC_GROUPREADERROR_IN Structure

```
typedef struct{
unsigned char ucEnable;
}MMC_GROUPREADERROR_IN;
```

Parameters

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_GROUPREADERROR_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned short usGroupErrorID;
}MMC_GROUPREADERROR_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

usGroupErrorID

Returned command group error ID. Signals where an group error has occurred within function block. These values are vendor specific. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Displays an error code integer.

Figure 5-113 describes the function block for MMC_GroupReadError

Figure 5-118: MMC_GroupReadError function block

5.9.6.2. Function Block Code Example

```
int rc;
MMC_GROUPREADERROR_IN       stGroupReadError_in;
MMC_GROUPREADERROR_OUT      stGroupReadError_out;
//
// Inserting the structure parameters:
stGroupReadError_in.ucEnable    = 1;      //Enabled
//
rc = MMC_GroupReadError (hConn, iAxisRef, &stGroupReadError_in, &stGroupReadError_out);
printf("Group Error[%ld] ErrId[%d]\n", (long int)stGroupReadError_out.usGroupErrorID,
(short)stGroupReadError_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

5.9.7. MMC_GroupReadStatus

For multiple Axes. Returns the status of an axes group according to the active Group function block. This is an administrative function block, since no movement is generated.

```
MMC_LIB_API int MMC_GroupReadStatusCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPREADSTATUS_IN* pInParam,
OUT MMC_GROUPREADSTATUS_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPREADSTATUS_IN** input data structure using the MMC_GroupReadStatus function.

pOutParam

Points to the **MMC_GROUPREADSTATUS_OUT** output structure receiving information, as a result of calling the MMC_GroupReadStatus function.

Remarks

The outputs reflect the commanded state of the group.

Scope

All

MMC_GROUPREADSTATUS_IN Structure

```
typedef struct{
unsigned int uiHndlr;
unsigned char ucEnable;
}MMC_GROUPREADSTATUS_IN;
```

Parameters

uiHndlr

Requested group handle. Values of any integer accepted.

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_GROUPREADSTATUS_OUT Structure

```
typedef struct{
unsigned long ulState;
unsigned short usStatus;
short usErrorID;
unsigned short usGroupErrorID;
}MMC_GROUPREADSTATUS_OUT;
```

Parameters

ulState

Group current state. Returned command status.

Refer to the sub-section **5.4 Axis Status** for the range of axis status bit masks with bitwise values.

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

usGroupErrorID

Returned command group error ID. Signals where an group error has occurred within function block. These values are vendor specific. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Displays an error code integer.

Figure 5-114 describes the function block for MMC_GroupReadStatus

Figure 5-119: MMC_GroupReadStatus function block

5.9.7.2. Function Block Code Example

```
int rc;
MMC_GROUPREADSTATUS_IN      stGroupReadStatus_in;
MMC_GROUPREADSTATUS_OUT     stGroupReadStatus_out;
//
// Inserting the structure parameters:
stGroupReadStatus_in.uiHndlr    = 35;     // Returned function block handle
stGroupReadStatus_in.ucEnable   = 1;      //Enabled
//
rc = MMC_GroupReadStatusCmd (hConn, iAxisRef, &stGroupReadStatus_in, &stGroupReadStatus_out);
printf("Group Status[%ld] ErrId[%d]\n", (long int)stGroupReadStatus_out.usGroupErrorID,
(short)stGroupReadStatus_out.usErrorID);
printf("Group Status[%ld]\n", (long int)stGroupReadStatus_out.ulState);
if (rc != 0)
{
    HandleError();
}
```

5.9.8. MMC_GroupReset

For multiple Axes. Makes the transition from the state GroupErrorStop to GroupDisabled by resetting all internal group-related errors – it does not affect the output of the function block instances.

```
MMC_LIB_API int MMC_GroupResetCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPRESET_IN* pInParam,
OUT MMC_GROUPRESET_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPRESET_IN** input data structure using the MMC_GroupReset function.

pOutParam

Points to the **MMC_GROUPRESET_OUT** output structure receiving information, as a result of calling the MMC_GroupReset function.

Remarks

This function block also resets all axes in this group like MMC_Reset.

Scope

All

MMC_GROUPRESET_IN Structure

```
typedef struct{
unsigned char ucExecute;
}MMC_GROUPRESET_IN;
```

Parameters

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_GROUPRESET_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_GROUPRESET_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-115 describes the function block for MMC_GroupReset

Figure 5-120: MMC_GroupReset function block

5.9.8.2. Function Block Code Example

```
int rc;
MMC_GROUPRESET_IN      stGroupReset_in;
MMC_GROUPRESET_OUT     stGroupReset_out;
//
// Inserting the structure parameters:
stGroupReset_in.ucExecute  = 1;    // Start the execution command

//
rc = MMC_GroupResetCmd (hConn, iAxisRef, &stGroupReset_in, &stGroupReset_out);
if (rc != 0)
{
    HandleError();
}
```

5.9.9. MMC_GroupSetOverride

For multi-axes. Sets the values of override for the coordinated motion of several axes, and all functions operating on that axes group.

```
MMC_LIB_API int MMC_GroupSetOverrideCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_SETOVERRIDE_IN* pInParam,
OUT MMC_SETOVERRIDE_OUT* pOutParam
);
```

Motion Mode NC - Supported Distributed – Not supported

Source **Function**
GMAS\includes\MMC_PLCopen_single_API.h

Structures

GMAS\includes\MMC_PLCopen_group_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_SETOVERRIDE_IN** input data structure using the MMC_SetOverride function.

pOutParam

Points to the **MMC_SETOVERRIDE_OUT** output structure receiving information as a result of calling the MMC_SetOverride function.

Remarks

The override parameters act as a factor that is multiplied to the commanded vector velocity, acceleration, deceleration, and jerk of all axes group function blocks. MMC_GroupSetOverride is not applicable to master-slave group synchronized motion commands.

MMC_GroupSetOverride does not influence the state diagram of the axis group. The following should be noted:

- The Input *fAccFactor* acts on positive and negative acceleration (deceleration).
- This function block sets the override factor, which is valid until a new override is set.
- The default values of the override factor are 1.0. The value of the overrides can be between 0.0 and 1.0. The behavior of values > 1.0 is vendor specific. Values < 0.0 are not allowed. The value 0.0 is not

allowed for both *fAccFactor* and *fJerkFactor*, and generates an error.

- The value 0.0 set to the *fVelFactor* stops the axis without bringing it to the state GroupStandby.
- Override does not act on slave axes groups in the state Group Synchronized motion.
- *fVelFactor* can be changed at any time and acts directly on the ongoing motion.
- Reducing the *fAccFactor* and/or *fJerkFactor* can lead to a position overshoot – a possible cause of damage.

Scope

All

MMC_SETOVERRIDE_IN Structure

```
typedef struct{
float fVelFactor;
float fAccFactor;
float fJerkFactor;
unsigned short usUpdateVelFactorIdx;
unsigned char ucEnable;
}MMC_SETOVERRIDE_IN;
```

Parameters

fVelFactor

New override factor for the velocity. Any +ve float value between [0 – 1].

fAccFactor

New override factor for the acceleration/deceleration. Any +ve float value between [0 – 1].

fJerkFactor

New override factor for the jerk. Any +ve float value between [0 – 1].

usUpdateVelFactorIdx

Index of changed velocity factor. Vendor defined. The default is 0. Has integer values of 0 - 2

This variable is not in use at this moment.

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_SETOVERRIDE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
} MMC_SETOVERRIDE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-116 describes the function block for MMC_GroupSetOverride

Figure 5-121: MMC_GroupSetOverride function block

5.9.9.2. Function Block Code Example

```
int rc;
MMC_SETOVERRIDE_IN          stGSetOverride_in;
MMC_SETOVERRIDE_OUT         stGSetOverride_out;
//
// Inserting the structure parameters:
stGSetOverride_in.fAccFactor            = 0.4;   //New override factor for the
acceleration/deceleration
stGSetOverride_in.fJerkFactor           = 0.1;   //New override factor for the jerk
stGSetOverride_in.usUpdateVelFactorIdx  = 1;     //Index of changed velocity factor
stGSetOverride_in.fVelFactor            = 0.4;   //New override factor for the velocity
stGSetOverride_in.ucEnable              = 1;     //Enabled
//
rc = MMC_GroupSetOverrideCmd (hConn, iAxisRef, &stGSetOverride_in, &stGSetOverride_out);
if (rc != 0)
{
    HandleError();
}
```

5.9.9.3. Implementation Example

Figure 5-122: Timing diagram for MMC_GroupSetOverride – Example

1. Axes Group Vector Velocity changes to 50% with 100% of deceleration
2. Axes Group Vector Velocity back to 100% with 50% acceleration
3. Axes Group Vector Velocity moves to 0% with 100% deceleration
4. No Change, because *fAccFactor* 0.0 is not allowed; Error is set

5.9.10. MMC_GroupSetPosition

For multiple Axes. Sets the Position of all axes in a group without moving the axes.

```
MMC_LIB_API int MMC_GroupSetPositionCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GROUPSETPOSITION_IN* pInParam,
OUT MMC_GROUPSETPOSITION_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed – Not supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GROUPSETPOSITION_IN** input data structure using the MMC_GroupSetPosition function.

pOutParam

Points to the **MMC_GROUPSETPOSITION_OUT** output structure receiving information, as a result of calling the MMC_GroupSetPosition function.

Remarks

The new coordinates are described in an array. Within the coordinate system input, the respective coordinate system is selected as a method of referencing or a transformation. MMC_GroupSetPosition shifts the position of the addressed coordinate system and affects the higher level coordinate systems (so if ACS is selected, MCS and PCS are affected).

Scope

All

MMC_GROUPSETPOSITION_IN Structure

```
typedef struct{
double dbPosition[NC_MAX_NUM_AXES_IN_NODE];
MC_COORD_SYSTEM_ENUM eCoordSystem;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
unsigned char ucMode;
}MMC_GROUPSETPOSITION_IN;
```

Parameters

dbPosition

Target position for the motion of the axis when conditions are met. Any -ve or +ve double values in technical unit [u].

Array of coordinates, incl. positions and orientations (Distance if Mode = RELATIVE). The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eBufferMode

MC_BufferMode defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE = 1

MC_BUFFERED_MODE = 2

MC_BLENDING_LOW_MODE = 3

MC_BLENDING_PREVIOUS_MODE = 4

MC_BLENDING_NEXT_MODE = 5

MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with

	the lowest velocity of both commands (1 and 2) at the first end-position (1).
BlendingPrevious	Blending with the velocity of function block 1 at the end-position of this block
BlendingNext	Blending with the velocity of function block 2 at end-position of function block1
BlendingHigh	Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucMode

RELATIVE =True, ABSOLUTE = False (Default)

RELATIVE means that Position is added to the actual position value of the axis at the time of execution. This results in a recalibration by a specified distance. ABSOLUTE means that the actual position value of the axis is set to the value specified in the Position parameter.

Values accepted are Boolean, TRUE/FALSE.

ucPosMode

Boolean values for the position mode can be absolute mode = 0, or relative mode = 1

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_GROUPSETPOSITION_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_GROUPSETPOSITION_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-118 describes the function block for MMC_GroupSetPosition

Figure 5-123: MMC_GroupSetPosition function block

5.9.10.2. Function Block Code Example

```
int rc;
MMC_GROUPSETPOSITION_IN    stGroupSetPos_in;
MMC_GROUPSETPOSITION_OUT   stGroupSetPos_out;
//
// Inserting the structure parameters:
stGroupSetPos_in.eCordSystem    = MC_MCS_COORD; //Define types of supported coordinate systems
stGroupSetPos_in.eBufferMode    = MC_BUFFERED_MODE;//MC_BUFFERED_MODE_ENUM Defines the behavior
of the axis
stGroupSetPos_in.dbPosition[2]  = 80000.0;//Array of coordinates, positions and orientations
stGroupSetPos_in.dbPosition[3]  = 50000.0;
stGroupSetPos_in.dbPosition[4]  = 60000.0;
stGroupSetPos_in.ucMode         = 1;//Position is added to the actual position value of the
axis
stGroupSetPos_in.ucExecute      = 1;
//
rc = MMC_GroupSetPositionCmd (hConn, iAxisRef, &stGroupSetPos_in, &stGroupSetPos_out);
if (rc != 0)
{
    HandleError();
}
```

5.9.11. MMC_RemoveAxisFromGroup

For multiple Axes. Removes one axis from the group AxesGroup. This is an administrative function block, since no movement is generated.

```
MMC_LIB_API int MMC_RemoveAxisFromGroup(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_REMOVEAXISFROMGROUP_IN* pInParam,
OUT MMC_REMOVEAXISFROMGROUP_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_REMOVEAXISFROMGROUP_IN** input data structure using the MMC_RemoveAxisFromGroup function.

pOutParam

Points to the **MMC_REMOVEAXISFROMGROUP_OUT** output structure receiving information, as a result of calling the MMC_RemoveAxisFromGroup function.

Remarks

The command cannot be buffered. If there is no axis remaining in the group, the state automatically changes to GroupDisabled.

Scope

Only available if the group is **disabled**. If issued on a group that is in GroupEnabled, GroupStandby, or GroupErrorStop state, it generates an error and the function block is not executed. If GroupEnabled, an explicit error is output:

NC_UNSUITABLE_NODE_STATE

MMC_REMOVEAXISFROMGROUP_IN Structure

```
typedef struct{
NC_IDENT_IN_GROUP_ENUM eIdentInGroup;
}MMC_REMOVEAXISFROMGROUP_IN;
```

Parameters

eIdentInGroup

The NC_IDENT_IN_GROUP_ENUM enumerator identifies the order and Nodes in the group of the added axis. Performed via an enumerator to give the different axes a name in the order, which can be coupled to the names in the kinematic model. The options are:

NC_NODE_1_ID = 0

……

NC_NODE_16_ID = 15

MMC_REMOVEAXISFROMGROUP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_REMOVEAXISFROMGROUP_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-119 describes the function block for MMC_RemoveAxisFromGroup

Figure 5-124: MMC_RemoveAxisFromGroup function block

5.9.11.2. Function Block Code Example

```
int rc;
MMC_REMOVEAXISFROMGROUP_IN      stRemoveAxisFromGrp_in;
MMC_REMOVEAXISFROMGROUP_OUT     stRemoveAxisFromGrp_out;
//
// Inserting the structure parameters:
stRemoveAxisFromGrp_in.eIdentInGroup     = NC_NODE_4_ID;//Identifies the supported axes (nodes)
in a group
//
rc = MMC_RemoveAxisFromGroup (hConn, iAxisRef, &stRemoveAxisFromGrp_in,
&stRemoveAxisFromGrp_out);
if (rc != 0)
{
    HandleError();
}
```

5.9.12. MMC_SetKinTransform

Sets a kinematic transformation between the ACS and MCS based on the predefined kinematic model for multi-axes. Refer to the section **5.8.1 Coordinate System and kinematic transformation on page 5-238** for a further detailed explanation.

```
MMC_LIB_API int MMC_SetKinTransform(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_SETKINTRANSFORM_IN* pInParam,
OUT MMC_SETKINTRANSFORM_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed – Not supported
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_SETKINTRANSFORM_IN** input data structure using the MMC_SetKinTransform function.

pOutParam

Points to the **MMC_SETKINTRANSFORM_OUT** output structure receiving information, as a result of calling the MMC_SetKinTransform function.

Remarks

A kinematic transformation is a representation of the machine construction. The input MMC_SETKINTRANSFORM_IN refers to a kinematic model including the parameters.

This function block always behaves on a pre-defined AxesGroup. Since a kinematic transformation always has to fit to an appropriate *AxesGroup*, a call to some group function block in MCS mode will lead to an error unless an appropriate *Kinematic* is defined. For a further detailed explanation, refer to the section **5.8.1 Coordinate System and kinematic transformation on page 5-238.**

Scope

The SetKinTransform function can only be called in the following states:

- Group_Standby
- Group_Disabled

Calling the function in any other results in the Error **-182**.

MMC_SETKINTRANSFORM_IN Structure

```
typedef struct{
double ulTrCoef[NC_MAX_NUM_AXES_IN_NODE][NC_MAX_NUM_COEF];
int iNumAxes;
MC_BUFFERED_MODE_ENUM eBufferMode;
NC_TR_FUNC_ID_ENUM iMcsToAcsFuncID[NC_MAX_NUM_AXES_IN_NODE];
NC_NODE_HNDL_T hNode[NC_MAX_NUM_AXES_IN_NODE];
NC_AXIS_IN_GROUP_TYPE_ENUM eType[NC_MAX_NUM_AXES_IN_NODE];
unsigned char ucExecute;
}MMC_SETKINTRANSFORM_IN;
```

Parameters

ulTrCoef

The parameter *ulTrCoef* is defined by the type of supported MCS transform function according to the definition of the array of [NC_TR_FUNC_ID_ENUM] enumerators defined by the name iMcsToAcsFuncID. The value of *ulTrCoef* is 8 bytes (double) and ranges between any ±15 digit value.

These coefficients depend on the transition function type. Presently, Elmo supports only two functions defined by the following enumerator NC_TR_FUNC_ID_ENUM:

NC_TR_NONE_FUNC = 0

NC_TR_SHIFT_FUNC = 1

Each of the above enumerations has a different connotation dependant on the transition function type. The coefficients are a two dimensional double array 16 x 3, with the following definitions:

16 – the maximum number of axes

3 – the three inputs for each axis

An important point when entering the coefficients:

In order to avoid problematic position adjustments (when using MCS mode) always maintain the following ratio between the coefficients:

NC_BACK_TR_RATIO_COEF x NC_FRWD_TR_RATIO_COEF =1

[NC_MAX_NUM_COEF] is defined as 3.

For further details refer to the section **5.8.1.4 Using the function MMC_SetKinTransform on page 5-242.**

iNumAxes

Number of axes. *iNumAxes* can have values of any positive integer [1….16].

eBufferMode

MC_BufferMode defines the behavior of the axis. Enumerator modes are as follows, but only the Aborting Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2

MC_BLENDING_LOW_MODE = 3
MC_BLENDING_PREVIOUS_MODE = 4
MC_BLENDING_NEXT_MODE = 5
MC_BLENDING_HIGH_MODE = 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

iMcsToAcsFuncID

Function ID to change between the MCS and ACS modes.

i*McsToAcsFuncID* can have one of the NC_TR_FUNC_ID_ENUM enumerator values 0 – 1. The size of the i*McsToAcsFuncID* is 16, the maximum number of members in the group.

The array parameter NC_MAX_NUM_AXES_IN_NODE is equal to 16, and defined as the maximum number of axes in a group.

hNode

NC_NODE_HNDL_T defines the Node references array. The array parameter NC_MAX_NUM_AXES_IN_NODE is equal to 16, and defined as the maximum number of axes in a group. Since this is in essence the axis reference, refer to the function in section **10.3.17 MMC_GetAxisByName.**

eType

NC_AXIS_IN_GROUP_TYPE_ENUM define the types of supported axis in the group. There are 16 possible values for the parameter *eType*:

NC_X_AXIS_TYPE = 0
NC_Y_AXIS_TYPE
NC_Z_AXIS_TYPE
NC_U_AXIS_TYPE
NC_V_AXIS_TYPE
NC_W_AXIS_TYPE
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
NC_S_AXIS_TYPE

The array parameter NC_MAX_NUM_AXES_IN_NODE is equal to 16, and defined as the maximum number of axes in a group.

ucExecute

For this function, this parameter is not relevant.

MMC_SETKINTRANSFORM_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SETKINTRANSFORM_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-120 describes the function block for MMC_SetKinTransform

Figure 5-125: MMC_SetKinTransform function block

5.9.12.2. Function Block Code Example

```
int rc;
MMC_SETKINTRANSFORM_IN      stSetKinTransform_in;
MMC_SETKINTRANSFORM_OUT     stSetKinTransform_out;
//
// Inserting the structure parameters:
stSetKinTransform_in.eBufferMode         = MC_BUFFERED_MODE;// MC_BUFFERED_MODE_ENUM Defines
the behavior of the axis
stSetKinTransform_in.ulTrCoef[2][5]      = 1; // Array of coordinates, incl. positions and
orientations
stSetKinTransform_in.ulTrCoef[3][5]      = 1;
stSetKinTransform_in.ulTrCoef[4][5]      = 1;
stSetKinTransform_in.iMcsToAcsFuncID[2]  = 2; //MCS to ACS function ID
stSetKinTransform_in.iMcsToAcsFuncID[3]  = 3;
stSetKinTransform_in.iMcsToAcsFuncID[4]  = 4;
stSetKinTransform_in.iNumAxes            = 6;// Position is added to the actual position value
of the axis
stSetKinTransform_in.ucExecute           = 1;
//
rc = MMC_SetKinTransform (hConn, iAxisRef, &stSetKinTransform_in, &stSetKinTransform_out) ;
if (rc != 0)
{
    HandleError();
}
```

5.9.12.3. Implementation Example

5.9.13. MMC_GroupReadParameter

Reads a specific group axes parameter.

```
MMC_LIB_API int MMC_GroupReadParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READPARAMETER_IN* pInParam,
OUT MMC_READPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Irrelevant	Distributed – Irrelevant
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READPARAMETER_IN** input data structure using the MMC_GroupReadParameter function.

pOutParam

Points to the **MMC_READPARAMETER_OUT** output structure receiving information as a result of calling the MMC_GroupReadParameter function.

Remarks

N/A

Scope

None

MMC_READPARAMETER_IN Structure

```
typedef struct{
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_READPARAMETER_IN;
```

Parameters

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READPARAMETER_OUT Structure

```
typedef struct{
double dbValue;
unsigned short usStatus;
short usErrorID;
}MMC_READPARAMETER_OUT;
```

Parameters

dbValue

Output of the specific parameter. Any Double value.

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-124 describes the function block for MMC_GroupReadParameter

Figure 5-126: MMC_GroupReadParameter function block

5.9.13.2. Function Block Code Example

5.9.14. MMC_GroupReadBoolParameter

Reads a specific group axes Boolean parameter.

```
MMC_LIB_API int MMC_GroupWriteParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READBOOLPARAMETER_IN* pInParam,
OUT MMC_READBOOLPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Irrelevant	Distributed – Irrelevant
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READBOOLPARAMETER_IN** input data structure using the MMC_GroupReadBoolParameter function.

pOutParam

Points to the **MMC_READBOOLPARAMETER_OUT** output structure receiving information as a result of calling the MMC_GroupReadBoolParameter function.

Remarks

N/A

Scope

None

MMC_READBOOLPARAMETER_IN Structure

```
typedef struct{
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_READBOOLPARAMETER_IN;
```

Parameters

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_READBOOLPARAMETER_OUT Structure

```
typedef struct{
long lValue;
unsigned short usStatus;
short usErrorID;
}MMC_READBOOLPARAMETER_OUT;
```

Parameters

lValue

Boolean parameters integer value

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-122 describes the function block for MMC_GroupReadBoolParameter

Figure 5-127: MMC_GroupReadBoolParameter function block

5.9.14.2. Function Block Code Example

5.9.15. MMC_GroupWriteParameter

Modifies the value of a specific group axes parameter.

```
MMC_LIB_API int MMC_GroupWriteParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_WRITEPARAMETER_IN* pInParam,
OUT MMC_WRITEPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Irrelevant	Distributed – Irrelevant
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEPARAMETER_IN** input data structure using the MMC_GroupWriteParameter function.

pOutParam

Points to the **MMC_WRITEPARAMETER_OUT** output structure receiving information, as a result of calling the MMC_GroupWriteParameter function.

Remarks

N/A

Scope

None

MMC_WRITEPARAMETER_IN Structure

```
typedef struct{
double dbValue;
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_WRITEPARAMETER_IN;
```

Parameters

dbValue

Parameter value. Any positive double (8 bytes)value up to 15 digits.

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_WRITEPARAMETER_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEPARAMETER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-123 describes the function block for MMC_GroupWriteParameter

Figure 5-128: MMC_GroupWriteParameter function block

5.9.15.2. Function Block Code Example

5.9.16. MMC_GroupWriteBoolParameter

Modifies the value of a specific group axes Boolean parameter.

```
MMC_LIB_API int MMC_GroupWriteBoolParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_WRITEBOOLPARAMETER_IN* pInParam,
OUT MMC_WRITEBOOLPARAMETER_OUT* pOutParam
);
```

Motion Mode	NC - Irrelevant	Distributed – Irrelevant
Source	GMAS\includes\MMC_PLCopen_group_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEBOOLPARAMETER_IN** input data structure using the MMC_GroupWriteBoolParameter function.

pOutParam

Points to the **MMC_WRITEBOOLPARAMETER_OUT** output structure receiving information, as a result of calling the MMC_GroupWriteBoolParameter function.

Remarks

N/A

Scope

None

MMC_WRITEBOOLPARAMETER_IN Structure

```
typedef struct{
long lValue;
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterArrIndex;
unsigned char ucEnable;
}MMC_WRITEBOOLPARAMETER_IN;
```

Parameters

lValue

Input value. Any +ve integer.

eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterArrIndex

Array index parameter. Any +ve integer values

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_WRITEBOOLPARAMETER_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEBOOLPARAMETER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 5-124 describes the function block for MMC_GroupWriteBoolParameter

Figure 5-129: MMC_GroupWriteBoolParameter function block

5.9.16.2. Function Block Code Example

Chapter 6: Error Handling

The function blocks provide an internal basic error checking on the input data, which is implementation dependent. For instance, if MaxVelocity is set to 6000, and the Velocity input to a function block is set to 10,000, then either the system slows down or an error is generated. In the case where an intelligent servo drive is coupled via a network to the system, the MaxVelocity parameter is probably stored on the servo drive. The function block provides for and handles the error generated by the drive internally. With another implementation, the MaxVelocity value could be stored locally. In this case, the function block will generate the error locally.

Both centralized and decentralized error handling methods are possible when using the motion control function blocks.

Centralized error handling is used to simplify programming of the function block. Error-reaction is the same independent of the instance in which the error has occurred.

Figure 6-1: Centralized error handling

Decentralized error handling allows for different reactions depending on the function block in which an error occurred.

Figure 6-2: Decentralized error handling

6.1. Buffered Errors

All buffered commands will abort if the applicable axis moves to the state ErrorStop. The Error output of the relevantly aborted function blocks are SET. Any subsequent commands will be rejected and the error output is SET.

If a function block has an error e.g. due to a wrong set of parameters, the error output is set, and the behavior is depending on the application program. For instance, with two function blocks, the first instance FB1 executes any motion command on an axis, and starts a new command on a second function block instance FB2 in buffered mode on the same axis. This command is buffered and waits until FB1 is completed (Done). Before the first instance FB1 has finished its command, let one of the following situations occur:

1. The axis goes to state ErrorStop (e.g. due to a following error or over-temperature). FB1 sets the output ERROR. FB2 (as well as any other function block instance that is waiting to execute a buffered command on this axis) sets its ERROR output and shows with the output ErrorID, that it cannot execute its job, because the axis is in a state that doesn't allow it. All buffered commands are cleared. After the axis error is reset by MMC_Reset, it can be commanded again.
2. The FB1 sets its Error output (e.g. due to an invalid parameterization). FB2 becomes active and executes the given command immediately afterwards, and the application should handle the error situation.

6.2. G-MAS Error IDs

The following table lists the G-MAS error IDs with an explanation for each error. These errors are a result of a continuous event check by the G-MAS, causing the system to halt its motion. The system enters an *Errorstop* state and must be Reset to continue its motion.

Error ID	Description	Resolution
0	Indicates that everything is OK, no errors and warnings	
-1	NC driver error	Critical error!!! Refer to Elmo for support.
-2	NC driver memory mapping failure	Critical error!!! Close all resources and restart the G-MAS.
-3	Record length is larger than the maximal recorder buffer size. Alternatively, input arguments to the MMC_UploadDataCmd function are not valid.	Verify that NumberOfSignals*NumberOfRows < 1,000,000. Verify the following legacy of the "From " and "To " parameters on MMC_UploadDataCmd function: 1. "From " < "To ". 2. ("To " - "From ") <= 1400. 3. "To " < RecordLen[byte]
-4	Illegal Record Gap value. Space between two successive records in units of CycleTime.	Verify that Recording Gap in the MMC_BEGIN_RECORDING_IN structure is greater than zero.
-5	There is a call to begin recording when the Recorder is already operating	Wait until the recording completes or stop the recording process using MMC_StopRecordingCmd function.
-6	Incorrect Node handle (axis or group of axes)	Verify that you are using the correct axis reference. If you are using the recording mechanism, verify that Recording group input and Recording Parameter input on the MMC_BEGIN_RECORDING_IN structure is correctly defined. If you use the SetKinematicTransformation function, verify that "NC_NODE_HNDL_T" array is correctly defined in the MMC_SETKINTRANSFORM_IN structure.
-7	The Node's function blocks' list is empty	The problem is in the function block list management. Refer to Elmo for support and please describe the exact scenario where it occurred.
-8	Node state is unsuitable for the active function block	You are trying to perform an illegal node state transition. Check that your drives are connected to the G-MAS. Refer to the allowed transitions in the PLCOpen state machine definition.
-9	Node is not found	This problem is in the Resource file. Please check that your Resource file is correct.

Error ID	Description	Resolution
		If correct, then check and validate that the axis name and ID is the same as in your application.
-10	You are trying to perform an operation that cannot operate from Disabled mode	To see the allowed node state, refer to the PLCOpen state machine definition.
-11	No free Nodes in the list	Be careful that you have not added more than 96 nodes to the Active Nodes list
-12	The node type is incorrect	You are trying to use an inappropriate node type: 1. You may use only SingleAxis (0) or MultiAxis (1) types. 2. Using a function that is related to single axis, only use a SingleAxis node type. 3. Using a function that is related to multi axis only use a MultiAxis node type.
-13	Incorrect function block type parameter	You are trying to use a incorrect function block number. Allowed numbers are [1…80]. Refer to Elmo for help.
-14	Free function blocks list is empty	You have entered the maximum number of function blocks to the node function block list. The maximum size is 1000 function blocks. Wait a delay time to allow the function blocks to execute. This will result in freeing the active function blocks list. To get the current function block depth use the MMC_GetFbDepthCmd function.
-15	Function block type not supported by G-MAS	You tried to use a function that is not presently supported by G-MAS. If you still wish to use this function, refer to Elmo for support.
-16	Function block pointer not found	Check if you have defined any nodes in the Resource file before operating with nodes. This error can be caused by an internal function block list management problem. Refer to Elmo for support.
-18	The function block Handle validity check failed	This is an internal problem. Refer to Elmo for support.
-19	The function block is already removed from the active list and marked as free	When the function block has completed its operation, it is automatically removed from the list and marked as free. If you work in IEC mode, it remains on the list even after the function block execution. To remove it from the list, run the **MMC_CmdStatus** function after the function block has completed its action.
-21	Axis used by other group.	You cannot use the same axis in two different groups or in a group and as a single axis node.

Error ID	Description	Resolution
		You can define the same axis in two different places. However, when the axis is activated from one place, it cannot be activated from another place.
-22	One of the parameters is out of the permitted range	If you receive this message while running the motion of a single axis function block, please check the following parameters: 1. 0 < Velocity <= 100E9 (for move velocity -100E9 <= Velocity <= 100E9). 2. 0 < Acceleration <= 1000E9. 3. 0 < Deceleration <= 1000E9. 4. 0 < Jerk <= 2E12. 5. 0 < Buffer Mode < 7. 6. 0 < Direction < 5. For each of the following functions being used, check the apprpriate parameter values: **Multi-axis motion function, check the parameters above** (for a single axis) and in addition check the parameters: 1. 0 <= CoordSystem < 4. 2. 0 <= Circle Mode <= 4. 3. 0 <= PathChoice < 3. 4. 0 <= Transition Mode < 6. **SetOverride function, check the parameters:** 1. 0 <= Velocity Factor <= 1. 2. 0 < Velocity Factor eNum < 3. **ReadDigitalInput function, check the parameter:** 0 < Input Number < 32. **PowerON\OFF function, check the parameter:** 1 <= Buffer Mode <= 2. **HomingDS402 function, check the parameter:** 1 <= Homing method <= 35. **To change the coordinate system between two consequential function blocks**, use the specific buffer mode MC_BUFFERED_MODE. The Move Path function supported transition mode is TM_NONE_MODE **When setting the Kinematic transformation function**, check the parameters: 1. NumAxes <= Actual number of axis on the group. 2. McsToAcsFuncID == NC_TR_SHIFT_FUNC (1). **Resexportfile or Resimportfile function, check the following parameter:**

Error ID	Description	Resolution
		0 <= DownloadType <= 3. **Recorder function with one of the following trigger types:** 1. TG_RECORDING_TRIGGER_TYPE_EDGE_WindowIn 2. TG_RECORDING_TRIGGER_TYPE_EDGE_WindowOut 3. TG_RECORDING_TRIGGER_TYPE_LEVEL_WindowInside 4. TG_RECORDING_TRIGGER_TYPE_LEVEL_WindowOutside. The uiRP.TG_RECORDING_TRIGGER_LEVEL_1 must be lower than uiRP.TG_RECORDING_TRIGGER_LEVEL_2. **ErrorCorrectionTable functions, check the parameters:** 1. 0 <= ErrorTableNumber <= 3 2. 0 <= TableResolution (axis grid size) <= 22 **HeartBeatTime functions, check the following parameter:** 0 < HeartBeatTime <= 65535 **CanSetSyncTime functions, check the following parameter:** SyncTime < 1000
-23	Axis Update Cycle Period or Shift is not compatible for the group	Verify that cycle period and cycle shift are equal in both, Group and axis.
-24	Incorrect coordinate system was used or the coordinate system is not enabled at the present	Use group function with an accepted and proper coordinate system defined, allowed parameters is: MC_ACS_COORD(1) MC_MCS_COORD(2) If you are using one of the coordinate systems above and still obtain this error then it siugnifies that this current function block does not support this coordinate system. For example; when using the MoveCircularAbsolute function, only MCS coordinate system is supported.
-25	Group cannot receive motion command because it's disabled	Run the Group Enabled function before sending motion commands
-26	Inappropriate connection status	To Download Firmware you need to be connected with only one RPC connection and without any IPC connections. Make sure that you connect to target G-MAS with only one EAS application (only one PC). Make sure that you do not have any active application in the G-MAS, if you have, stop the prosess or reset the G-MAS. Make sure that you do not have any application execution in the StartUp script at: mnt/jffs/usr/UserStartUp.sh.
-28	Cannot open a UDP socket	Check that the socket is not already open. Check that the connection handler is correct, and that the function loads.

Error ID	Description	Resolution
-29	Open UDP socket fail	Refer to Elmo for support.
-30	Bind UDP socket fail	Refer to Elmo for support.
-31	Calling this function from an IPC connection is not permitted	This function cannot operate in IPC connection mode. Run this function in RPC connection mode.
-32	Incorrect connection type parameter	The connection type parameter of the node is not permitted for this action. For example, If you are trying to download firmware, only perform it through the RPC connection. If the error continues, refer to Elmo for support.
-33	Cannot perform any of the operations listed in the Resolution when at least one of the axis is powered on	If you wish to operate one of the following functions, please verify that all your axes are powered Off. 1. Download the file through TFTP. 2. Download the firmware. 3. Exit the G-MAS application. 4. Enable a communication gateway. 5. Enable G-MAS config mode (preop mode).
-34	Update of the UDP socket on-connection table fails	Cannot update the UDP socket table due to an incorrect connection index or error in the table. Refer to Elmo for support.
-35	Update of the UDP address on-connection table fails	Cannot update the UDP address table due to input error or error in the table. Refer to Elmo for support.
-36	Update of the event mask on-connection table fails	Cannot update the event mask table due to incorrect connection index or error in the table. Make sure that the requested mask is not negative. Otherwise, refer to Elmo for support.
-37	Get UDP socket from connection table fails	Cannot get the UDP socket with the given index. Check the index boundaries. Verify that you really have a socket on this index. Refer to Elmo for support.
-39	Cannot read event mask	Refer to Elmo for support.
-40	Cannot operate this specific function in the current mode	There are certain functions that can only work in CONFIG_MODE_CONFIG. To change the mode from CONFIG_MODE_REGULAR to CONFIG_MODE_CONFIG run the MMC_ConfigCmd function. To return to CONFIG_MODE_REGULAR mode run MMC_ExitCmd" function.

Error ID	Description	Resolution
-43	Set DHCP function sent with the wrong input parameter	When you send the Set_DHCP function you must update the input MMC_DHCP_SET_IN structure. The structure contains one object, set according to the following specification: 1. 0 - Disable DHCP. 2. 1 - Enable DHCP.
-44	Internal process failure	Refer to Elmo for support.
-45	Failed to read the download version status from the flash params table	Refer to Elmo for support.
-46	Failed to recreate the configuration table	Refer to Elmo for support.
-47	Cannot open the directory where resource files are located	Check if the following directory exist /mnt/jffs/MMC/config/resources on G-MAS. If you do not have such a directory, create it and download a new resource file to the directory.
-48	Open file failed	Cannot open resource file, check that the resource file exists in the following path: /mnt/jffs/MMC/config/resources/MMCResources.xml. If yes, check the file is correct. If no, create it using EAS and download a new resource file to the directory.
-49	Motion parameter assignment failed	Assign only one of the following motion types to the function: 1. eCAN_OPEN_MOVE_VELOCITY 2. eCAN_OPEN_MOVE_ABSOLUTE 3. eCAN_OPEN_MOVE_RELATIVE 4. eCAN_OPEN_MOVE_STOP 5. eCAN_OPEN_MOVE_HALT 6. eCAN_OPEN_MOVE_HOME
-50	Node in Distributed mode	In Distributed mode, you cannot load a new motion function block when the axis is currently in motion. Wait until the motion is complete and resend the function.
-51	Homing failed	Increase the delay to the HomeTimeOut value. If this does not solve the problem, try to simulate the same homing scenario without G-MAS.
-52	Distributed base processor failure	Refer to Elmo for support.

Error ID	Description	Resolution
-53	G-MAS does not support this function or this working mode (NC\Distributed)	Try to change the profile mode. If you are in NC mode, change to Distributed mode. If you are in Distributed mode, change to NC mode. If this does not solve the problem, refer to Elmo for support.
-54	Group contains less than 2 axes.	You cannot perform the Group Enable command if you have less than two axes in the group. You can add axes to a group in two ways: 1. Create a new resource file using EAS that has a group with more than one axis inside and download the resource file to G-MAS. 2. Use MMC_AddAxisToGroup function to add axis to group dynamically during the running of the application.
-55	Connection handler is incorrect	Critical error!!! You sent an incorrect connection handler to the function. Check that the connection handler is not zero. Check that you perform the connection process successfully, and then use the correct connection handler.
-56	Incorrect motion operation mode	When operating in Distributed mode you must set the appropriate operation mode for each movement function block type: 1. Move Abs\Rel\Add function must work in position profile operation mode. 2. Move velocity\continues function must work in Velocity profile operation mode. 3. Move homing function must work in Homing operation mode. If you try to enable group, all the group members should be in proper NC motion mode. Change the operation mode using the MMC_ChngOpMode function.
-57	The communication type is not appropriate to this function	There are functions for specific connection types (CAN\EtherCAT). If you do not have a proper connection type, you cannot run this function. Try to find another alternative or refer to Elmo for support.
-58	Wrong function argument type	The inputs to the function are not as expected in the API definition. Go to the API function block definition, find the function and verify that all the inputs are correct.
-59	Group is not found	Check that the name of the group in the resource file is the

Error ID	Description	Resolution
		same as in the input structure of this function. Resource file location - "/mnt/jffs/MMC/config/resources/ " Input structure name - "MMC_AXISBYNAME_IN " Function name - "MMC_GetGroupByNameCmd "
-60	Network scan failure	Critical error!!! The Network Scan function must only work with CAN network. Check the hardware connections between the G-MAS and the drive. Also, make sure to place terminators in the empty sockets. Verify that the drive's are powered on. Verify that your resource file is appropriate for the setup. If after this validation, the error persists, refer to Elmo for support.
-61	Network get statistics failure	Critical error!!! The GetCommStatistics function only operates with EtherCAT network. 1. Check the hardware connections between the G-MAS and the drive(s). 2. Verify that the servo drives are power on. 3. Verify that your resource file is appropriate to the setup. 4. Verify that you have proper EtherCAT network configuration file located on the G-MAS at: "/mnt/jffs/MMC/config/Ethercat/cfg.xml" - you can create this file using the EAS application. If after this validation the GetCommStatistics function still does not operate, refer to Elmo for support.
-62	Network reset statistics failure	The ResetCommStatistics function only operates with EtherCAT network: 1. Check the hardware connections between the G-MAS and the drive's. 2. Verify that the servo drives are powered on. 3. Verify that your resource file is appropriate to the setup. 4. Verify that you have proper EtherCAT network configuration file that located on the G-MAS here: "/mnt/jffs/MMC/config/Ethercat/cfg.xml" - you can create this file using the EAS application. If after this validation, the ResetCommStatistics function still does not operate, refer to Elmo for support.
-63	FS(File System) failure.	Critical error!!! Refer to Elmo for support.

Error ID	Description	Resolution
-64	Dynamic memory allocation fail	Critical error!!! Refer to Elmo for support.
-65	XML parser failure	Critical error!!! Check that all of the following files exist: 1. /var/MMC/config/resources/MMCResources.xml 2. /var/MMC/config/parameters/MMCGeneralPrm.xml 3. /var/MMC/config/parameters/<NODE Name #1>.xml 4. /var/MMC/config/parameters/<NODE Name # Number of nodes>.xml 5. /var/MMC/config/parameters/<group Name #1>.xml 6. /var/MMC/config/parameters/<NODE Name #Number of groups>.xml
-66	Open communication failure.	Critical error!!! Check that you are not already connected. Check that the PC and G-MAS are sufficiently connected (check ping between them). If you working with EtherCAT, check that, the cfg.xml is created and located at: /mnt/jffs/MMC/config/Ethercat/cfg.xml and is defined properly.
-67	Close communication failure.	Check if you are not already connected.
-68	Communication scan bus process failure	This function relevant only for CAN network. Refer to Elmo for support.
-69	An attempt was made to add an axis member to a group when the axis or ID is already present in the group.	Try to change the ID of the axis you wish to add, to an unused ID. If you still receive the error, then this node handle is already in the group.
-70	Max number of members in group is reached	Do not add more than 16 axes to a group. If you use the MMC_RemoveAxisFromGroup function, do not use an index larger than 16 to remove the axis.
-72	Profiler recalc end velocity failure	Critical error!!! Refer to Elmo for support.
-74	Real time driver not initialized	Critical error!!! You have a serious problem with your resource file. Check the following parameters: 1. Maximum number of nodes must be less than 96. 2. **For EtherCAT** a. Minimal cycle time is 1000[usec] = 1[msec]. b. Maximal cycle time is 100,000[usec] = 100[msec].

Error ID	Description	Resolution
		c. Minimal MB cycle time is 1000[usec] = 1[msec]. d. Maximal MB cycle time is 200,000[usec] = 200[msec]. e. Minimal BG cycle time is 100[usec] = 0.1[msec]. f. Maximal BG cycle time is 2000[usec] = 2[msec]. g. CycleTime <= MB CycleTime. h. MB CycleTime <= BG CycleTime * 1000. 3. **For CAN** a. Minimal cycle time is 1000[usec] = 1[msec]. b. The baud rate must be one of the following * 125 * 250 * 500 * 800 * 1000. 4. Maximum number of nodes in a group is 16. 5. Missing NC_AXIS definition of non-group type. 6. Missing MODULO definition of non-group type. 7. Missing DIVIDER definition of non-group type. 8. The Type of each node in the "TYPE" field must be one of the following: a. "ds301 " b. "ds401 " c. "ds402 " d. "ds406 “ e. "virtual " 9. The connection type in the "NET_TYPE " field must be one of the following: a. "CAN " b. "EtherCAT " 10. If you use the /var/MMC/config/parameters/MMCGeneralPrm.xml file to update global parameters, take care to set their low and high limits correctly. 11. DIVIDER value must be in the following range [0...4]. 12. MODULO value must be in the following range [0..(DIVIDER-1)].
-75	Wrong parameter cast type	Refer to Elmo for support.
-76	Write attempt to Read Only parameter	You cannot change this parameter. Refer to the G-MAS Administrative and Motion API. If you have any further questions refer to Elmo for support.

Error ID	Description	Resolution
-77	Incorrect transformation function	When you running MMC_SetKinTransform function you should set the transformation function type as input to the MMC_SETKINTRANSFORM_IN structure. Only the single NC_TR_SHIFT_FUNC (1) function is supported by the G-MAS. Set only this function. For further support, contact Elmo.
-78	Error in opening Real time driver	Critical error!!! Refer to Elmo for support.
-79	Internal process failure	Refer to Elmo for support.
-80	Internal process failure	Refer to Elmo for support.
-81	Incorrect member node indexed	When you wish to remove a node from the group, you must use valid indexes. The allowed range is [0…15].
-82	Group object creation failure	The group creation failed. Check the resource file on the G-MAS located at: /mnt/jffs/MMC/config/resources/MMCResources.xml. 1. Verify that the group name is unique. 2. Verify that the group ID is unique. 3. Verify that all the named Members of the group are defined with their complimentary Node name. 4. Verify that the DIM parameter is same as the number of Members. 5. Verify that the number of Members is less than 16. 6. Verify that the TYPE is "group ".
-83	Resource is not a group	Refer to Elmo for support.
-84	Resource is an incorrect communication type	Open the resource file on the G-MAS, located at: /mnt/jffs/MMC/config/resources/MMCResources.xml Verify that the NET_TYPE is one of the following: 1. CANbus 2. ETHERCAT If they are not, set the NET_TYPE correctly and restart the G-MAS.

Error ID	Description	Resolution
-86	Resource is an incorrect node type	The Type of each node in the "TYPE" field must be one of the following: a. "ds301 " b. "ds401 " c. "ds402 " d. "ds406 " e. "virtual ".
-87	Resource is set with an incorrect cycle time	Check the CycleTime configurations in the resource file located at: /mnt/jffs/MMC/config/resources/MMCResources.xml. **For EtherCAT:** a. Minimal cycle time is 1000[usec] = 1[msec] b. Maximal cycle time is 100,000[usec] = 100[msec] c. Minimal MB cycle time is 1000[usec] = 1[msec] d. Maximal MB cycle time is 200,000[usec] = 200[msec] e. Minimal BG cycle time is 100[usec] = 0.1[msec] f. Maximal BG cycle time is 2000[usec] = 2[msec] g. CycleTime <= MB CycleTime h. MB CycleTime <= BG CycleTime * 1000. **For CAN:** a. Minimal cycle time is 1000[usec] = 1[msec]
-88	Resource is using the incorrect baudrate	This error is only relevant for CAN connections. Verify that the baud rate in the resource file is one of the following: * 125 * 250 * 500 * 800. * 1000
-89	Parameter file is out of range	If you use the following file: /var/MMC/config/parameters/MMCGeneralPrm.xml as input for Axis Parameters to the G-MAS, make sure to set their low and high limits correctly. To see the limits, refer to the Axis Parameter table.
-90	Resource maximum number of nodes reached	The maximum number of axes has been reached. Reduce the axes number to 96.
-92	Resource file has an incorrect node cycle period divider	The DIVIDER value must be in the following range [0...4].

Error ID	Description	Resolution
-93	Resource file has an incorrect node cycle shift modulo	The MODULO value must be in the following range [0..(DIVIDER-1)].
-94	The resource tried to create a node object and failed	The problem is in the resource file. Check the following parameters: 1. Maximum number of nodes must be less than 96. **2. For EtherCAT** a. Minimal cycle time is 1000[usec] = 1[msec] b. Maximal cycle time is 100,000[usec] = 100[msec] c. Minimal MB cycle time is 1000[usec] = 1[msec] d. Maximal MB cycle time is 200,000[usec] = 200[msec] e. Minimal BG cycle time is 100[usec] = 0.1[msec] f. Maximal BG cycle time is 2000[usec] = 2[msec] g. CycleTime <= MB CycleTime h. MB CycleTime <= BG CycleTime * 1000. 3. **For CAN** a. Minimal cycle time is 1000[usec] = 1[msec]. b. The baud rate must be one of the following. * 125 * 250 * 500 * 800 * 1000. 4. Maximum number of nodes in a group is 16. 5. Missing NC_AXIS definition of non-group type. 6. Missing MODULO definition of non-group type. 7. Missing DIVIDER definition of non-group type. 8. The Type of each node in the "TYPE" field must be one of the following: a. "ds301 " b. "ds401 " c. "ds402 " d. "ds406 " e. "virtual " 9. The connection type in the "NET_TYPE " field must be one of the following: a. "CAN " b. "EtherCAT " 10. If you use the /var/MMC/config/parameters/MMCGeneralPrm.xml file to update global parameters take care about their low

Error ID	Description	Resolution
		and high limits. 11. DIVIDER value must be in the following range [0...4]. 12. MODULO value must be in the following range [0..(DIVIDER-1)].
-95	Resource tried to create a group object and failed	Refer to Elmo for support.
-96	Resource has reached the maximum number of nodes in the group	Check in the Resource file, and make sure that you do not exceed the maximum limit of nodes in a Group. The maximum number of nodes in a group is 16.
-97	The parameters for the shared memory key failed	Refer to Elmo for support.
-99	The parameters for the Get shared memory, failed	Refer to Elmo for support.
-100	Communication initialization failure	Refer to Elmo for support.
-101	One group member is not in StandStill	You tried to move the Group's state to Enabled while its state is Disabled. One of the group members is not in StandStill state. Check the state of all axes in the group. For each axis that is in Disabled state, run the Power ON command using the MMC_PowerCmd function. If an axis is in error state, run the MMC_ResetCmd function to reset the error and then run the Power ON command. If an axis is in motion state, wait until it completes the motion before running the Group Enable function.
-103	Synchronized motion is prohibited	You are not able to operate any motion function such as MoveAbsolute, MoveRelative, MoveVelocity, Stop, Halt etc.to a single axis drive, when the same drive is a member of a group whose state is enabled or any other state other than GroupDisabled. To operate motion of this axis you have the following possible solutions: * Disable the group where it is a member and then run the single axis motion function. * Remove this axis from all groups where it is a member, and then run the single axis motion function. * Move this axis using a MultiAxis functions like MoveLinearAbsoulute, MoveLinearRelative, GroupStop, GroupHalt etc.
-104	Incorrect recording parameter	You exceeded the maximum signal number (signal index). When you define which signal to record in the Recording

Error ID	Description	Resolution
		Vector, the maximum signal number can only be 57.
-105	Change of coordinate system while in motion is not permitted	You cannot change the coordinate system when the group is in motion. Wait until the motion is completed, and then change the coordinate system. Use the MMC_GetFbDepthCmd function to examine how many function blocks are currently in the list. If you receive "0 ", there are no function blocks in the list and you can change the coordinate system.
-106	MCS coordinate system is not set	If you try to operate a function in the MCS coordinate system, you must to run the MMC_SetKinTransform function before executing the new function. If you perform any changes in your group members (add\remove axis) you must run the MMC_SetKinTransform function again.
-108	The motion is not 2D circular angle motion	You tried to operate the MoveCircularAbsolute function in Angle mode but the number of axis in your group is more than 2. Use MoveCircularAbsolute function with only 2 axes in the group.
-109	Parameter index has an incorrect value	When using Read\Write Real\Bool parameter functions, the parameter index must be limited to [0…51].
-110	Get diagnostics has a communication failure	Critical error!!! Check your cable connections. Verify that you configured the EtherCAT network on the configuration file properly. The EtherCAT configuration file must be located at: /mnt/jffs/MMC/config/Ethercat/cfg.xml If you still get this error, refer to Elmo for support.
-111	Parameter has no value	When you using Read\Write Real\Bool parameter functions, the parameter value contain NULL. Refer to Elmo for support.
-112	Internal process failure	Refer to Elmo for support.
-113	Already in desired mode	If you want to Enable\Disable one of the following: * CommGateWay * ConfigMode But you cannot, you are already in the desired state.
-114	Internal process failure	Refer to Elmo for support.

Error ID	Description	Resolution
-115	You have chosen an incompatible Buffer mode vs. Transition mode combination	If you use a group motion function, you must enter a Buffer mode and Transition mode.Not all combinations are permitted. The list of all permitted combinations: 1. MC_ABORTING_MODE & MC_TM_NONE_MODE 2. MC_BUFFERED_MODE & MC_TM_NONE_MODE 3. MC_BLENDING_LOW_MODE & MC_TM_DEFINED_VELOCITY_MODE 4. MC_BLENDING_LOW_MODE & MC_TM_CORNER_DISTANCE_MODE 5. MC_BLENDING_LOW_MODE & MC_TM_MAX_CORNER_DEVIATION_MODE 6. MC_BLENDING_LOW_MODE & MC_TM_SWITCH_RADIUS_MODE 7. MC_BLENDING_PREVIOUS_MODE & MC_TM_DEFINED_VELOCITY_MODE 8. MC_BLENDING_PREVIOUS_MODE & MC_TM_CORNER_DISTANCE_MODE 9. MC_BLENDING_PREVIOUS_MODE & MC_TM_MAX_CORNER_DEVIATION_MODE 10. MC_BLENDING_PREVIOUS_MODE & MC_TM_SWITCH_RADIUS_MODE 11. MC_BLENDING_NEXT_MODE & MC_TM_DEFINED_VELOCITY_MODE 12. MC_BLENDING_NEXT_MODE & MC_TM_CORNER_DISTANCE_MODE 13. MC_BLENDING_NEXT_MODE & MC_TM_MAX_CORNER_DEVIATION_MODE 14. MC_BLENDING_NEXT_MODE & MC_TM_SWITCH_RADIUS_MODE 15. MC_BLENDING_HIGH_MODE & MC_TM_DEFINED_VELOCITY_MODE 16. MC_BLENDING_HIGH_MODE & MC_TM_CORNER_DISTANCE_MODE 17. MC_BLENDING_HIGH_MODE & MC_TM_MAX_CORNER_DEVIATION_MODE 18. MC_BLENDING_HIGH_MODE & MC_TM_SWITCH_RADIUS_MODE 19. MC_BUFFERED_MODE & MC_TM_SWITCH_RADIUS_MODE

6.2.1. Error correction error IDs

Error	Description	Resolution
-116	You have a problem with the header of the Error correction input file	Check that the error correction input file exists in the correct path you associated it. The error correction input file has a 5-rows header. Each row must contain a valid value according to the description: 1. Table size (number of points) 2. Start offset – the offset from the beginning of the shared memory segment 3. Error Table dimension (1, 2, or 3) 4. Start position - #Xpos #Ypos ... (as dimension number) 5. Axis grid size - Axis step on the correction table (in units $Log_2(X)$) (number of values as dimension number) 6. Table dimensions - #Row's #Column's
-117	The Error correction table is wrong	Check the data table in the file you created is correct. The size of the table must be as described in the header. All the values in the table must be numbers. All the delimiters between the numbers must be a tab.
-118	Cannot add another table	You have reached the maximum number of error correction tables. The maximum number of tables is 4. If you wish to add to this table, remove another table using DisableErrorTable and then the UnloadErrorTable functions. Another reason for this error maybe that you have reached the maximum limit of values in all ErrorCorrectionTables. The maximum limit is 100,000.
-119	Error Correction Table is already taken	You cannot allocate the same table twice.
-120	Cannot unload error correction table	You are trying to unload an error correction table that is already enabled. To unload the table run DisableErrorTable function and then run the UnloadErrorTable function.
-123	Cannot operate with current error correction table	You are trying to operate with an error correction table that is not loaded. Load the table using LoadErrorTable function and then try again.

-124	Error correction table is already mapped to this axis	You cannon allocate an error correction table to an axis when another error table is already allocated to this axis. If you wish to allocate this error correction table to this axis, free this axis of any other error correction tables using DiableErrorTable and UnloadErrorTable functions.

6.3. Continued G-MAS Error IDs

Error ID	Description	Resolution
-125	Cannot change axis operation mode	You cannot change the operation mode of the axis when the axis is a member of a group. First, remove this axis from all groups where it is a member, and then, run the function.
-126	You have exceeded the limit of the ModBUS table	Make sure that you have not exceeded the limit of any of the ModBUS tables. Their limits are as follows: Holding register - 31743 Coils - 1,024 Inputs - 1,024
-127	ModBUS table ID is incorrect	When referencing the ModBUS table, the table ID should be legal. The table ID should be an integer between [1 ... 247].
-128	The ModBUS server is already running	You cannot start the ModBUS server when the ModBUS is already in RUNNING state. To check the current ModBUS state use MMC_MbusIsRunning function. To change the ModBUS to non-RUNNING state, use MMC_MbusStopServer function.
-129	The file transfer through TFTP failed	**Download** Make sure that: Your TFTP server is turned on and points to the correct directory. The file name you enter is correct and the file exists. You set the correct IP address. **Upload** Make sure that: Your TFTP server is turned on and points to the correct directory. You set the correct IP address.

-130	Transition mode is out of range	Use only one of the following transition modes: 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 MC_TM_CORNER_DIST_CV_POLYNOM3 = 7 MC_TM_CORNER_DIST_CV_POLYNOM5 = 8
-131	The error table number is out of range	You can only use one of the following numbers: 0,1,2,3.
-132	Error table resolution is out of range	Set a permitted resolution between [1.....22].
-133	The current segment in the error table is out of range	Check the following parameters: * Start offset should be a positive number. * Start offset should be a less then 100,000. * (Error table size + Start offset) sould be less then 100,000.
-134	Trigger window parameters is out of range	You entered illegal recording parameters to the recording parameters vector (Rp). When you set the MMC_BEGIN_RECORDING_IN structure, make sure that the Rp[4] value are less or equal to Rp[5].
-135	The file type that you chose does not exist	When you download\upload file through TFTP you can only use one of the following file types: MMC_PARAMETER_FILE_DOWNLOAD = 0 MMC_RESOURCE_FILE_DOWNLOAD = 1 MMC_SNAPSHOT_RESOURCE_FILE_DOWNLOAD = 2 MMC_ETHERCAT_CFG_FILE_DOWNLOAD = 3 MMC_PERSONALITY_FILE_DOWNLOAD = 4
-136	Incorrect "SetIsToLoad" value	The input value uses a logical convention. You can only set [0\1] values as input.
-137	Set override input arguments are out of range	You have set the incorrect input arguments to the override function. Check the following inputs: * Velocity factor should be between 0.1 and 1. * Update the velocity factor Idx to only be NC_OVERRIDE_MAX_VEL_FACTOR (3).
-138	The number of axes is out of range	When using the "MMC_SetKinTransform" function, you cannot set the number of axes to larger than the actual number of axes in the group.

		The number of axes should be the actual number of axes that comprise the group motion.
-139	Transformation function is out of range	The transformation function can be only one of the following: * NC_TR_NONE_FUNC (0) – where there are no transitions between ACS and MSC. * NC_TR_SHIFT_FUNC (1) – where there are linear transformations between ACS to MCS but no reverse transformations between MCS and ACS.
-140	Buffer mode is out of range	You can only choose one of the following modes: MC_NULL_BUF_MODE = 0 MC_ABORTING_MODE = 1 MC_BUFFERED_MODE = 2 MC_BLENDING_LOW_MODE = 3 MC_BLENDING_PREVIOUS_MODE = 4 MC_BLENDING_NEXT_MODE = 5 MC_BLENDING_HIGH_MODE = 6
-141	Homing method is out of range	The homing methods range is [1 .... 35]
-142	Digital input number is out of range	The input digital range is [0 .... 31]
-143	Maximum velocity parameter is out of range	Check the maximum allowed velocity using the "MMC_ReadParameter" function. The maximum velocity parameter enumerator is "MMC_MAX_VELOCITY_PARAM" (15).
-144	Maximum acceleration parameter is out of range	Check the maximum allowed acceleration using the "MMC_ReadParameter" function. The maximum acceleration parameter enumerator is "MMC_MAX_ACCELERATION_PARAM" (19).
-145	Maximum deceleration parameter is out of range	Check the maximum allowed deceleration using the "MMC_ReadParameter" function. The maximum deceleration parameter enumerator is "MMC_MAX_DECELERATION_PARAM" (23).
-146	Maximum jerk parameter is out of range	Check the maximum allowed jerk using the "MMC_ReadParameter" function. The maximum jerk parameter enumerator is "MMC_MAX_JERK_PARAM" (26).
-147	Direction mode is out of range	The direction mode can be only one of the following: MC_POSITIVE_DIRECTION = 1 MC_SHORTEST_WAY = 2 Not implemented as yet MC_NEGATIVE_DIRECTION = 3 MC_CURRENT_DIRECTION = 4
-148	Zero or negative velocity input	Change the velocity parameter to a positive value.

-149	Coordinate system is out of range	The coordinate system can be only one of the following: MC_ACS_COORD = 1 MC_MCS_COORD = 2 MC_PCS_COORD = 3
-150	Circle mode is out of range	The circle mode can be only one of the following: MC_BORDER_CIRC_MODE= 1 MC_CENTER_CIRC_MODE = 2 MC_RADIUS_CIRC_MODE = 3 MC_ANGLE_CIRC_MODE = 4
-151	Path choice is out of range	The path choice can be only one of the following: MC_NONE_PATH_CHOICE = 0 MC_CLOCKWISE = 1 MC_COUNTERCLOCKWISE = 2
-152	Arc short long is out of range	TThe arc short long can only be one of the following: MC_NONE_ARC_CHOICE = 0 MC_SHORT = 1 MC_LONG = 2
-153	Input position is out of range	The maximum allowed position defined by Modulu parameter in the SetPosition function.
-154	Unable to set the requested slave state	Before beginning the FOE procedure, make sure to set the G-MAS and change all slaves automatically to BOOT state. At the end of the FOE procedure, make sure to reset the G-MAS and change all the slaves automatically back to OPERATIONAL state. If one of the slaves fails to change, state you will receive this error. Try changing the slave state manually using the EtherCAT configurator in EAS. Otherwise, Refer to Elmo for support.
-155	FoE File system error	Make sure that the requested file is successfully copied to the G-MAS and located at: “/tmp/[file name]”. Check that the file permissions are set to “rw”. Otherwise, Refer to Elmo for support.
-156	Write FoE packet error	Refer to Elmo for support.
-157	Unable to close FoE communication	Refer to Elmo for support.
-158	You have reach the maximum number of FoE retries	Verify that you have performed the bulk read configuration successfully, before reading the data.

-159	Failure during FoE TFTP transfer	Make sure that the input arguments of the DownloadFoE function are correct. The server should be set to PC IP. The filename should also be present in the PC.
-160	The G-MAS unable to open FoE connection with any one of the slaves	Check the communication cables between the G-MAS and drives. Otherwise, Refer to Elmo for support.
-161	Not in use at this time	Not in use at this time
-162	Cannot retrieve the recorded Bulk Read values array	Refer to Elmo for support.
-163	Wrong input to ConfigBulkRead function	Check the following parameters: ucIsPreset should be set to 0 or 1. uBulkReadParams-> eBulkReadPreset should be set between 1 and 4.
-164	You have exceeded the maximum data packet size	The maximum packet size is 1400 [bytes]. To reduce the packet size, reduce one of the following parameters: Number of axes Number of signals for each axis
-165	Bulk read memory allocation failed	Refer to Elmo for support.
-166	Trying to read from an unallocated memory segment	Verify that you have performed the bulk read configuration successfully, before reading the data.
-167	Number of axes is out of range	The number of axes can be between 1 and 100.
-168	Configuration enumerator is out of range	The function bulk read can work separately and parallel for two different applications.
-169	Unable to create FoE Download Thread	Refer to Elmo for support.
-170	EtherCAT system function failure	Check the communication cables between the G-MAS and drives. Otherwise, refer to Elmo for support.
-171	There is no open EtherCAT communication on the G-MAS	Refer to Elmo for support.
-172	You have entered an incorrect number of slaves	The number of slaves should be greater than zero and less or equal to the actual number of slaves
-173	You have entered the wrong slave ID to the input structure	The input structure of the function contains the list of slave ID's. Each of them should be valid, and should reference an equal or less than, but greater than zero, actual number of slaves.

-174	Wrong input enumerator to IEC control function	Set a legal enumerator value to the input structure
-175	IEC system error	Refer to Elmo for support.
-176	The present type of parameter is not suitable for this function	If you are using the following: [read\write] Axis Parameter function — Only use axis related parameters [read\write] Group Parameter function — Only use group related parameters [read\write] Global Parameter function — Only use global related parameters
-177	Cannot retrieve the node pointer	Make sure that you have entered a correct Axis\Group reference.
-178	The node is in motion	You cannot clear the function block list while the Axis\Group is in motion. Stop the axis\group and repeat the function command.
-179	Unable to clear the function block list	Refer to Elmo for support.
-180	Transition mode is not supported	Please use another transition mode.
-181	Wrong coefficient ratio	When using the NC_TR_SHIFT_FUNC function in a Kinematic transformation function, always maintain the ratio between the coefficients (actual used in the axes): NC_BACK_TR_RATIO_COEF•NC_BACK_SHIFT_COEF = 1 The maximum permitted deviation is 1E-5
-182	Inappropriate state for the SetKinTransform function	The SetKinTransform function can only operate in one of the following states: 1. GROUP_DISABLED 2. GROUP_STANDBY
-183	Illegal input value to MMC_SetEnquireFbStatusCmd	The input variable will not accept all numerical values (including -1, and 2). It will only accept 0 or 1.
-184	The path choice input is not acceptable	Please verify that the path choice parameter is correctly set for the circle mode you are using: • Border mode - Must be MC_NONE_PATH_CHOICE • Center mode - Must be MC_NONE_PATH_CHOICE • Radius mode - Must not be MC_NONE_PATH_CHOICE • Angle mode - Must be MC_NONE_PATH_CHOICE
-185	The arc short long input is not acceptable	Please verify that the arc short long parameter is correctly set for the circle mode you are using: • Border mode - Must be MC_NONE_ARC_CHOICE • Center mode - Must not be MC_NONE_ARC_CHOICE • Radius mode - Must not be MC_NONE_ARC_CHOICE

		• Angle mode - Must be MC_NONE_ARC_CHOICE
-186	**General profiler error**	**This is an emergency error. Call Elmo immediately.**
-187	An incorrect event mode input was set to the function	Please reset the input Event mode to any of the following: (0) - No notification (1) - Cycle notification (2) - Immediate notification
-188	This function is only supported in a 3D situation	If you are using the circular radius function, define the system kinematics (using MMC_SetKinTransform) so that it includes all Cartesian directions (X,Y,Z)
-189	Cannot initiate another bulk upload process	Only one bulk upload process is allowed at a time. Make sure that the current bulk process has completed using "MMC_GetBulkUploadStatusCmd", and then try again.
-190	A message arrival has timed out	Refer to Elmo for support.
-191	The response message is unexpected when considering the request message (DS301 layer).	Refer to Elmo for support.
-192	Failed to create the thread managing the upload process	Refer to Elmo for support.
-193	Bulk upload initialization process failed	Refer to Elmo for support.
-194	Internal resources allocation error	Refer to Elmo for support.
-195	The delay setting between messages failed	Refer to Elmo for support.
-196	Trying to upload a buffer that is zero or negatively sized	Refer to Elmo for support.
-197	Trying to upload a buffer that is larger than the IPC packet size	Make sure that the high buffer limit is lower than 65,600.
-198	No pending messages in the queue	Refer to Elmo for support.
-199	No valid message – zero length	Refer to Elmo for support.
-200	Unexpected message – communication layer.	Refer to Elmo for support.
-201	The sequence number is unexpected	Refer to Elmo for support.
-202	Illegal bulk upload buffer limits	Make sure that the high buffer limit is higher that low buffer limit. Make sure that the difference between high and low limit, is lower than 1400 (IPC packet size).
-203	The current kinematic direction is	Make sure that you only select a supported kinematic direction.

	not supported	
-204	N axes definition are not consecutive	When you define the kinematic directions in the vector, make sure that all used N axes are defined consecutively.
-205	The personality file format is wrong	Make sure that the personality file located at: var/MMC/config/resources/Personality.xml is not corrupt.
-206	The requested data is not present in the personality file	Make sure that the personality file located at: /var/MMC/config/resources/Personality.xml is up to date.
-207	Not all the axes are defined in the Kinematic group	When you perform a kinematic definition you should set all the members of the group to the kinematic group.
-208	Unsupported definition of only one Cartesian direction	Elmo does not support the Cartesian coordinate system in only one direction. If you wish to continue using the Cartesian coordinate system in only one direction: Either simply run the linear FB's in ACS mode. The disadvantage of using ACS mode is that you cannot use the transformation mechanism. Alternatively, another solution is to use this axis as N axis.
-209	Trying to select an unsupported coordinated system e.g. PCS	Currently, only the following coordinate systems are supported: • ACS. • MCS. Error generated is NC_NOT_SUPPORTED_COORDINATE_SYSTEM
-210	Current transition mode is not supported in ACS mode	When inserting an FB in ACS mode,only set one type of transition mode. The only supported transition mode in ACS is "MC_TM_CORNER_DIST_CV_POLYNOM5_NAXES"
-211	Contradiction between high and low software limits	When setting the high or low software limit, make sure that the "low" limit is lower than the "high" limit and vice versa.
-212	Motion with the current function block is forbidden with the present hardware limit	When the axis is located at a hardware limit, the only motion permitted is using MC_MoveAbsolute\Relative or other permitted function blocks.
-213	Motion with current function block is forbidden at the ACS software limit	When the axis is located at a software limit, the only motion permitted is using MC_Move[Linear]Absolute\Relative or other permitted function blocks.
-214	Multi Axis motion in the current direction is forbidden due to the presence of an ACS limit	When at least, one of the axes in the group is located at an ACS software limit, the only motion allowed is when the direction of each problematic axis is opposite to the direction of the limit: * software high limit – negative * hardware FLS limit – negative * software low limit – positive * hardware RLS limit – positive

		Refer to the section 3.2. Status Register in the G-MAS Administrative and Motion API.
-215	Motion with current function block is forbidden at the MCS software limit	When at least, one of the kinematic directions of the group is located at a software limit, the only motion permitted is using MC_MoveLinearAbsolute\Relative or other permitted function blocks.
-216	Single Axis motion towards a software limit is forbidden	Check the software limit values of the current axis using the MC_ReadParameter function with the following enumerators: * MMC_software_LIMIT_HIGH_POS_PARAM - for high limit * MMC_software_LIMIT_LOW_POS_PARAM - for low limit Then make sure to request movement of the axis to positions only within the limits.
-217	Multi Axis motion toward a software limit is forbidden	Check the software limit values in all members of the current group: **When working in ACS mode** Use the MC_ReadParameter function, for each member with the following enumerators: * MMC_software_LIMIT_HIGH_POS_PARAM - for high limit * MMC_software_LIMIT_LOW_POS_PARAM - for low limit **When working in MCS mode** Use the MC_GroupReadParameter function (in group level) with the following enumerators: * MMC_MCS_HIGH_LIMIT_ARRAY - for high limit * MMC_MCS_LOW_LIMIT_ARRAY - for low limit Then make sure to request movement of the group only to positions where all members\directions are between these limits.
-218	Single Axis motion in the current direction is forbidden due to the presence of a limit	When the axis is located at a hardware\software limit, the only motion allowed is in the opposite direction to the limit: * software low limit – positive * software high limit - negative * hardware FLS limit – negative * software RLS limit – positive Refer to the section 3.2.3 32bits Status Register in the G-MAS Administrative and Motion API.
-219	Multi Axis motion to current direction is forbidden due to the presence of an MCS limit	When at least, one of the kinematic directions of the group are in MCS software limit, the only motion allowed is when the motion direction of each problematic kinematic direction opposes the direction of the limit: * software low limit - positive * software high limit – negative Refer to the section 3.2.3 32bits Status Register in the G-MAS

		Administrative and Motion API.

6.4. G-MAS PVT Motion Error IDs

Error ID	Description	Resolution
-221	Bulk upload send acknowledgement failed	Refer to Elmo for support
-222	Failed to open the file containing the PVT/ECAM table	Check that the file exists, and has the relevant permissions, etc.
-223	Problem in header of the PVT/ECAM file	Check that the file is compliant with the format Elmo supplied/Refer to Elmo for support
-224	Trying to execute ECAM motion on PVT table, or vice versa	Execute the PVT motion on the PVT table, or ECAM motion on the ECAM table
-225	Failed to read the table values	Check that the file is compliant with the format Elmo supplied/Refer to Elmo for support
-226	Failed to insert the PVT/ECAM table to the G-MAS RAM	Verify that the table size does not exceed the following limit: TableSize = (24 + Dimention*(12 + NumberOfPoints*16) + NumberOfPoints*8)[bytes] should be lower than 1[Mbyte]. Verify that you have unused memory entries. The maximum number of inserted table points is 20. If the number of inserted table points is >20, perform UnloadTable, and then reload the table pointss in table groups of 20.
-227	The actual dimension of the vector does not match the dimension mentioned in header or trying to execute a vector motion on a 1D table	Change the input parameters according to requirements
-228	The path ID provided is not legal	Provide a legal hMemHandle, as returned by MMC_LoadTableFromFileCmd() or MMC_InitTableCmd()
-229	Trying to move an unallocated segment	Use MMC_InitTableCmd() and MMC_AppendPointsToTableCmd()/MMC_LoadTableFromFileCmd()
-230	The read and expected number of points do not match	Provide a correct expected number of points in header /edit the table so that the values will match
-231	Trying to execute a PVT/ECAM motion on some axis/vector, but the path was prepared for another axis/vector	Use the correct axis/vector
-232	Provided a path that contains two or less points	Provide a longer path
-233	Journal handle out of range (similar to NC_PVT_ECAM_MEM_HANDLE_OUT_OF_RANGE)	Refer to the resolution of the error NC_PVT_ECAM_MEM_HANDLE_OUT_OF_RANGE

Error ID	Description	Resolution
-234	File name contains invalid characters	Provide a legal file name

6.5. Continued G-MAS Error IDs

ID	Explanation	Resolution
-235	Buffered\Blended function block insertion at a limit is forbidden	When the axis\group is at a limit, buffered\blended motion is forbidden. At the limit, insert any motion function block in abortion mode, or when the function block queue of the axis\group is empty.
-236	Torque out of range	Please set torque within the allowed range.
-237	Parameter index is out of range	Please use only accessible indexes.
-238	Cannot define the same node twice	When you define the kinematic of the system, assign each member in a group to a kinematic direction. However, the same member cannot be assigned twice. Make sure that no member is defined twice in the hNode array.
-239	Could not retrieve the current index	Refer to Elmo for support
-240	Trying to append data too close to the current index	Refer to Elmo for support
-241	Trying to append data to uninitialized table	Use MMC_InitTableCmd() prior to MMC_AppendPointsToTableCmd()
-242	Trying to append data beyond the end of the table/overlapping the current index	Provide a smaller segment to append
-243	The start index provided is beyond the end of table	Provide a legal start index
-244	The end index calculated is beyond the end of table	Provide a smaller segment to append
-245	No data to append	Provide data to append
-246	"Holes" inside the table	Append to index adjacent to previously appended
-247	Underflow threshold value is larger than the segment length	Underflow threshold value cannot be bigger than the segment length. Provide smaller underflow threshold value.
-248	Maximum Modbus packet size exceeded	You cannot request to read or write a packet larger than 250 memory cells at one function call.
-249	Axis Link mode is out of range.	You should set one of the following Link types : 0 – Target Mode 1 – Actual Mode
-250	One of the axes is not in StandStill state.	When linking two axes, both of them should be in StandStill state.

ID	Explanation	Resolution
-251	A slave Linked axis cannot be physical axis.	When linking two axes, the slave cannot be a physical axis. The slave axis should be a virtual axis, defined in the system resource file.
-252	The offset variable of the Slave Linked axis should equal zero.	When trying to link or unlink two axes, the offset variable of the slave axis should be zero. Send a motion command to the slave axis with zero target position before linking or unlinking the axis.
-253	Cannot unlink an axis when slave axis in motion.	When trying to unlink two axes, the slave axis should be in a non-motion state. If the axis is in motion, wait until the motion has completed and repeat the unlinking process. Otherwise, run the Stop procedure for the slave axis and repeat the unlinking process.
-254	Cannot link the same axis twice.	When the axis is already linked as a Master or Slave, it cannot be linked again. If you wish to link the current axis to some other axis, first unlink it and then retry the linking process.
-255	Axis Linking is only supported for DS402 nodes.	When trying to link two axes, both must be DS402 nodes.
-256	Cannot open parameters file	Cannot open parameters file, check that the parameters file exists in the following path: "/mnt/jffs/MMC/config/parameters/MMCParameters.xml". If yes, check the file is correct. If no, create it by using ""MMC_SaveParamCmd" function or copy a new file from PC using "MMC_ResImport" function or through EAS interface.
-257	Cannot use character variable for recording trigger	Please use other variable for trigger
-258	Current parameter cannot be part of queued write	Change this parameter in immediate mode
-259	Wrong number of parameters	The number of parameters should be between 1 and 5.
-260	Dwell parameter is small	The dwell parameter presented in microseconds and cannot be less than the cycle time.
-261	Maximum limit of Immediate function blocks is reached	When an immediate administrative function block is inserted, do not insert more than 5 FB's to same motion FB.
-262	Cyclic mode out of range	When working with PVT\ECAM, the sequence can be run in cyclic mode or in single mode. Choose the mode using "ucIsCyclic" parameter: 0 - non-cyclic mode

ID	Explanation	Resolution
		1 - cyclic mode
-263	Dynamic mode out of range	When working with PVT\ECAM, the sequence can be run in dynamic mode or static mode. In dynamic mode, you can add additional point on the fly. Choose the mode using "ucIsDynamic" parameter: 0 - static mode 1 - dynamic mode
-264	Auto mode out of range	When working with PVT\ECAM in Dynamc mode, you can select where to enter the additional points in the table. In auto mode the additional points will be inserted from the last inserted point. In manual mode the additional points will be inserted from the start index provided by user. Choose the mode using "ucIsAutoAppend" parameter: 0 - manual mode 1 - auto mode
-265	Position mode out of range	When working with PVT\ECAM, you can define whether the position in the file\table is absolute or relative to the last commanded position. Choose the mode using "ucIsPosAbsolute" parameter: 0 - relative mode 1 - absolute mode
-266	Time mode out of range	When working with PVT\ECAM, you can define whether the time of the each point in the file\table is absolute or relative to the last point. Choose the mode using the "ucIsTimeAbsolute" parameter: 0 - absolute mode 1 - relative mode
-267	File mode not allowed for this function	When working with PVT\ECAM, you should choose the type of the motion table "NC_MOTION_TABLE_TYPE_ENUM". If you trying to run one of the following functions: * MMC_InitTable * MMC_AppendPointsToTable The mode of the table cannot be a file type, only an array type. Set the table type using "eTableType" variable.
-268	Cannot append to static table	When working with PVT\ECAM, you can add points dynamically using the Append function. In order to provide the appendment, you should initialize the table in dynamic mode, using the "ucIsDynamic" variable.

ID	Explanation	Resolution
-269	Small time interval	When working with PVT\ECAM, the time interval between two consecutive points cannot be less than the cycle time.
-270	Spline insertion is forbidden	A spline table cannot be inserted when you already have a PVT\ECAM table in the memory. Remove the PVT\ECAM table using "MMC_UnloadTable" function and then retry.
-271	PVT\ECAM insertion is forbidden	A PVT\ECAM table cannot be inserted when you already have spline table in the memory. Remove the spline table using "MMC_PathUnselect" function and then retry.
-272	Cannot insert non-immediate administrative function block to pending node	When a motion function block is entered with execute "0", it will move the axis\group to the pending state. The axis will stay in this state until the user enters another motion function block with execute "1". In pending state, any non-immediate administrative function block insertion is forbidden. Make sure that the last motion function block prior to the non-immediate administrative function block, is inserted with execute "1".
-273	Cannot use absolute time	When you operate with PVT in cyclic mode, you cannot use absolute time. In cyclic mode set the IsAbsoluteTime flag to **0** in order to work in relative time mode.
-274	Cannot copy the parameters file to target destination	Refer to Elmo for support.
-275	Memory overlap between error correction tables	When you insert error correction table, specify the start offset where the table will be inserted. If the new insertion will cause a memory overlapping with another previously inserted error correction table, the new table will not insert and an error is generated. Remove the unnecessary tables from G-MAS and then retry to insert the table. Otherwise try to choose a better start offset.
-276	Cannot enable virtual encoder	In order to enable the virtual encoder two conditions should be fulfilled: * Virtual encoder in disabled * RPDO3 are not configured
-277	Contradiction between coordinate systems	When operating with Splines or Multi axis PVT, define the requested coordinate system (ACS\MCS\PCS) on both, table insertion and motion functions. If the coordinate system is different for the insertion and motion functions, an error is generated.

ID	Explanation	Resolution
		Select the same coordinate system in both functions.
-278	Support only Cartesian axes	When operating with Splines or Multi axis PVT in MCS mode, the only supported axes is the Cartesian axes, X, Y, Z. Run the function SetKinTransform with the requested vector and choose only Cartesian directions, with no "N" axes, Polar axes, or "S".
-279	Append block is too large	The maximum size of append block is 170 doubles. Maximum number of points is calculated as: MaxPoints = 170/[2*Dimention+1]. 1D - 56 points 2D - 35 points 3D - 24 points etc...
-280	Table size is too large	You are trying to insert too large a PVT\Spline table. Try to reduce the table size and try again.
-281	Node sent a CAN SDO abort reply	This error occurs after the GMAS sends an SDO download\upload request to the node and receives an SDO abort response. The response code is in accordance with the CiA SDO download/upload protocol found in the EtherCAT Application Manual section 6.2. Abort SDO Transfer Protocol. Please check that the destination object index and sub-index are supported by the CANopen standards or by the specific vendor standards.
-282	SDO request timeout	GMAS sent SDO download\upload request and did not receive any response in the default timeout period. Please check physical connections with the node and that it is powered on.
-283	CAN driver failure	CAN chip driver failure occurred, please refer to ELMO for support.
-284	Segmented SDO toggle bit unchanged	Segmented SDO toggle bit remained unchanged since last segment transfer. Please see DS301 CiA "application Layer and communication profile" document for further details. Refer to ELMO for support.
-285	Invalid PDO number	PDO number is invalid for this API, please see API documentation for allowed PDO numbers for this API (try 0,1 instead of 3,4).

ID	Explanation	Resolution
-286	Status word fault bit reset timeout	GMAS failed to reset the device status word fault bit in the default timeout period. Please check the reason for the device fault in the device manual and correct it.
-287	Motor off request from the drive Failed during drive initialization	Please check device for errors preventing it from changing to ds402 FSTM state SWITCH_ON_DISABLED, SWITCHED_ON or READY_TO_SWITCH_ON. Refer to ELMO for support
-288	Wrong PDO length	Invalid generic PDO length request, possible generic lengths range is 1-8 bytes.
-289	CAN error interrupt	CANbus error interrupt received from CAN chip. Please check the bus state for termination problems, or try to reduce the bus load. If no device is connected on the bus, then connect a device and check if GMAS recovered from fatal error state.
-290	Inconsistency between GMAS operation mode request and device actual state	Device failed to change operation mode in the default timeout period after GMAS change operation mode request. Please check device manual for possible reasons for this inconsistency
-291	Device interpreter returned Error response	Device interpreter returned response with error bit (6th bit of the 4th byte) set. See device documentation for reasons for this error.
-292	Invalid interpreter command length	Interpreter command length is invalid. For set and get commands, the maximal length is 8 bytes. For drive program execution commands the maximal length is 80 bytes
-293	UDP message send failed	Linux "send" command failed - please refer to ELMO for support
-294	Axis is virtual	Attempt to perform an operation that is incompatible with the virtual axis. To perform this operation, work with a real axis.
-295	Linux "strtol" command failed	Linux "strtol" command failed - please refer to Elmo for support
-296	Status word IP bit failed to zero	Status word IP bit failed to zero. Please refer to Elmo for support
-297	Resource file does not specify the IO type	Configuration resource file does not indicate what is the type of DS401 device connected to GMAS, please change your configuration.
-298	Invalid PDO group number	PDO group number is incompatible for this API, please see API documentation for allowed group numbers for this API

ID	Explanation	Resolution
-299	Another reset operation is in progress for this node	Attempt to perform reset operation on a node that is already in reset process
-300	GMAS failed to create telnet server thread	Operation system problem, please refer to ELMO for support
-301	Cannot connect between Table-Related function and Non Table-Related function types	There are two types of motion functions, Table-Related: * Spline * PVT * ECAM And Non-Table-Related: * Move[Linear]Absolute[Repetitive] * Move[Linear]Relative[Repetitive] * MoveCircularAbsolute * MovePolynomAbsolute * MoveVelocity When building the motion sequence, only use the same function block type without amalgamating them.
-302	Cannot connect two consequent polynomial functions blocks	The G-MAS does not support insertion of two consequent polynomial function blocks.
-303	Abort mode not supported	This function block does not support abort mode. To interrupt the motion use the Stop\GroupStop function and then insert the motion function block in buffered mode.
-304	Bulk Read: Not supported signal	When configuring bulk read using a non-preset mode, select the requested signals and place it in the input array: *stInput.uBulkReadParams.ulBulkReadParameters* Each entry in this array holds a requested signal. Setting a signal that is not supported by G-MAS will produce an error. Check which signals are supported by the G-MAS and only set the supported signals. **NOTE:** If there are unused entries, set their places to "0".
-306	Set override IDX is out of range	Set only "0" as the "usUpdateVelFactorIdx" variable.
-307	GMAS failed to create RPC server thread	Operation system problem, please refer to ELMO for support
-308	GMAS failed to create IPC server thread	Operation system problem, please refer to ELMO for support
-309	GMAS failed to schedule thread	Operation system problem, please refer to ELMO for support
-310	GMAS failed to open a socket	Operation system problem, please refer to ELMO for

ID	Explanation	Resolution
		support
-311	GMAS failed to socket change socket owner	Operation system problem, please refer to ELMO for support
-312	GMAS failed to change socket option	Operation system problem, please refer to ELMO for support
-313	GMAS failed to get IP address from socket	Operation system problem, please refer to ELMO for support
-314	GMAS failed to bind a socket	operation system problem, please refer to ELMO for support
-315	GMAS failed to perform listen on a socket	Operation system problem, please refer to ELMO for support
-316	GMAS failed to perform accept on a socket	Operation system problem, please refer to ELMO for support
-317	GMAS failed to create thread attribute	Operation system problem, please refer to ELMO for support
-318	GMAS failed to detach a thread	Operation system problem, please refer to ELMO for support
-319	GMAS failed to perform accept on a socket	Operation system problem, please refer to ELMO for support
-320	GMAS failed initialize IPC server	Please refer to ELMO for support
-321	the maximal number of connections reached	The maximal number of IPC\RPC connections has reached. Please try to close some of the connections
-322	GMAS failed open EtherCAT communication	Please refer to ELMO for support
-323	Device failed to change EtherCAT state after GMAS request in the default timeout period.	Please check device manual for possible reasons or refer to ELMO for support
-324	another node initialization is in progress	Node sent two boot - up messages, the latter of the two was discarded - please check that node initialized successfully.

6.6. NC Driver Warning IDs

ID	Explanation	Resolution
2000	Drive warning: The Axis is already in Power OFF mode	Only run the PowerOFF function when the axis is not in PowerOff state. To read the current status of the axis call the MMC_ReadStatusCmd function
2001	Drive warning: The Axis is already in Power ON mode	Only run the PowerON function when the axis is not in PowerOn state. To read the current status of the axis call the MMC_ReadStatusCmd function
2002	The Axis is already in current Operation mode	Do not run the "Change Operation Mode" function when the axis is presently active in the same operation mode.
2003	The Axis TouchProbe is already in Enabled state	Do not run the TouchProbe Enable function repeatedly.
2004	The Axis TouchProbe is already in Disabled state	Do not run TouchProbe Disable function repeatedly. When the touch probe is triggered, the state changes automatically to disabled state.

6.7. NC Profiler Error IDs

The following table lists the NC profiler error IDs. Many of these errors are a result of human programming errors, although some may be caused by limitations of the G-MAS itself. In most situations where an error of the type below is produced, it may be possible to overcome the error without the necessity to reset the system.

Error	Explanation	Resolution
-1000	Error to get active FB ptr	Refer to Elmo for support.
-1001	Error to get next FB ptr	Refer to Elmo for support.
-1002	Error to get previous FB ptr	Refer to Elmo for support.
-1003	Error to get last FB ptr	Refer to Elmo for support.
-1004	Error to get previous FB ptr on reverse flow	Refer to Elmo for support.
-1005	Error to get last FB ptr on reverse flow	Refer to Elmo for support.
-1006	Error to get next FB ptr on reverse flow	Refer to Elmo for support.
-1007	Error to get previous FB ptr (Reject MSEP)	Refer to Elmo for support.
-1008	Error to get next FB ptr (Reject MSEP)	Refer to Elmo for support.
-1010	Get active FB pointer rejected	Refer to Elmo for support.
-1011	Cannot get FB by number in the queue	Refer to Elmo for support.
-1012	The FB pointer that entered to profiler is NULL	Refer to Elmo for support.
-1026	3D circle mode is not defined properly	For circular movement functions. When using a MC_CENTER_CIRC_MODE circle mode, the circular motion "PI" or "2PI" (180 or 360 degrees) cannot be created.

Error	Explanation	Resolution
-1027	Contradiction between circle params	This error is caused by inputting improper parameters to the circular function. If you are using the circular radius function: The auxiliary point is a radius vector (value and direction). The radius vector must be orthogonal with a vector created between the start point and the end-point of the circle (circle surface). The maximal permitted deviation (result of dot product between radius vector and a circle surface) is 1E-2 [user units]. If you are using circular center function: The auxiliary point is a center point of the circle. The center point of a circle must be at the same distance from start and end-points, meaning, constant radius. The maximal permitted deviation of these distances is 1 [user unit]. NOTE: The definition of a circle is where each point on the perimeter of a circle is the same distance from the center.
-1028	Profiler Error: Transition mode is not supported	This function does not support any transition modes. Set the buffer mode to "MC_BUFFERED_MODE" and transition mode to "MC_TM_NONE_MODE"
-1029	Obtain negative value in CalcMaxPossibleVendOpt()	Refer to Elmo for support.
-1030	Profiler Error: Small distance between start and end point in Circular Radius mode	This error originates from the MoveCircular function in Radius mode. The length of the straight line between the start and end-points cannot be less than $1e^{-8}$ [user units] The length is defined by: $\sqrt{(\boldsymbol{End}[\mathbf{0}]-\boldsymbol{Start}[\mathbf{0}])^2+(\boldsymbol{End}[\mathbf{1}]-\boldsymbol{Start}[\mathbf{1}])^2+(\boldsymbol{End}[\mathbf{2}]-\boldsymbol{Start}[\mathbf{2}])^2}$
-1031	Cross product between radius and StartToEnd point vector is very small	This error originates from the MoveCircular function in Radius mode. The cross product between the radius vector defined in the auxiliary point, and the vector between the start and end-points are very small. The minimal permitted value for the cross product is $1e^{-9}$ [user units]. Increase the distance between the start and end-points, or increase the radius vector length.
-1032	Radius vector length is zero (Radius mode)	This error originates from the MoveCircular function in Radius mode.

Error	Explanation	Resolution
		The length entered for the Auxiliary point vector must be larger than $1e^{-9}$ [user units] To calculate the radius length use the formula: $\sqrt{\mathbf{Aux[0]^2 + Aux[1]^2 + Aux[2]^2}}$
-1034	The radius of the circle is very small (center or angle mode)	This error originates from the MoveCircular function in either Center or Angle mode. In these modes, when you enter the input parameters to the function, you are defining the end-point and the center of the circle point. You cannot enter parameters that create a circle with radius smaller than $1e^{-9}$ [user units] To calculate the radius of the circle use the formula: $\sqrt{(\mathbf{Aux[0] - Start[0]})^2 + (\mathbf{Aux[1] - Start[1]})^2 + (\mathbf{Aux[2] - Start[2]})^2}$
-1035	Cannot calculate two possible centers in circle radius mode	Refer to Elmo for support.
-1039	Zero max velocity on current FB	The G-MAS cannot operate a function block where the maximum velocity is zero. The current function block will cause no movement of an axis. Enter the movement function with a non-zero length movement.
-1040	Small Angle parameter	This error originates from the MoveCircular function in Angle mode. In this mode, the Angle input should be inserted as a value to the Aux[N] vector, in the "0" location. The units of the angle are degrees. For instance, if you wish to create a full circle, you should enter the following value: Aux[0] = 360; The minimum permitted absolute (can be negative) value (|x|) of the angle is 1 degree. Make sure you enter an angle with a value above the minimum limit.
-1041	Sweep angle equal to 180° or 360°	This error originates from the MoveCircular function in center mode. In this mode, circular motion cannot operate near to values of 180 or 360 degrees. The exact forbidden area is: * 180° ± 0.58° * 360° ± 0.58°

Error	Explanation	Resolution
-1043	Cannot operate Circle Angle function in non 2D mode	When using Circle Angle function, the exact number of axes in the group must be 2.
-1045	Profiler Error: Two points are very close	This error originates from the MoveCircular function in Border mode. In this mode three points are defined: * Start point SP * Border point BP * End point EP Prior to running the function block, check that each set of these points are not too close to each other (SP vs. BP, SP vs. EP, BP vs. EP). If all of the points are close together, an error will be created. The logic of close proximity checking: If((ABS(Point[1].x - Point[2].x) <$1e^{-7}$) && (ABS(Point[2].y - Point[3].y) <$1e^{-7}$) && (ABS(Point[1].z - Point[3].z) <$1e^{-7}$)) Then, the points are very close (not permitted) Else, the points are OK (permitted)
-1046	Three points on the same line	This error originates from the MoveCircular function in Border mode. In this mode, the three points Start point(SP), Border point(BP), and End point(EP), are defined. Prior to running the function block, make sure that the Start point(SP), Border point(BP), and End point(EP) are not located on the same straight line (two points define the line). If they are located on the same line, an error is created.
-1047	Profiler Error: Operate with empty vector	When trying to select a spline path to a specific vector, the vector should contain at least one member.
-1048	Profiler Error: Operate with a large vector	When trying to select a spline path to a specific vector, the vector should contain no more than 16 members.
-1050	Transition curve cannot be inserted in Line-Line mode - angle between two lines close to 0°	The profiler can only insert a transition curve between two lines when the angle between the lines is greater than 0.06°.
-1051	Transition curve cannot be inserted in angle between two lines close to 0°	The profiler can only insert a transition curve between two lines when the angle between the lines is greater than 0.06°.
-1052	Transition curve cannot be inserted in angle between two lines close to 180°	The profiler can only insert a transition curve between two lines when the angle between the lines is less than 179.94°.

Error	Explanation	Resolution
-1053	Line Circle transition curve not created	The end point of the first function block does not intersect with the start point of the second function block. Maximum permitted deviation between these two points is $1e^{-3}$ [user units]
-1054	Circle Line transition curve not created	The end-point of the first function block does not intersect with the start point of the second function block. Maximum permitted deviation between these two points is $1e^{-3}$ [user units]
-1055	Cannot create a "Switch Radius" transition curve between two Circles	The end-point of the first function block does not intersect with the start point of the second function block. Maximum permitted deviation between these two points is $1e^{-3}$ [user units]
-1056	Cannot create a Spline profile with a short length between points	Verify that the radius of the first Circle is not equal to the radius of the Switch Radius parameter (transition parameter). If these radii are equal, change the value of one of them. If not, refer to Elmo for support.
-1057	Cannot create a Spline profile with zero time execution between points	When using Spline profiles, make sure that the minimal length between two consecutive points is 1 [user units]. Make sure that the execution time between a movement of two consecutive points is greater than zero. Check that the segment length is not zero. The segment length is defined by: $\sqrt{(\mathrm{End}[0]-\mathrm{Start}[0])^2+(\mathrm{End}[1]-\mathrm{Start}[1])^2+(\mathrm{End}[2]-\mathrm{Start}[2])^2}$ It is recommended not to allow the velocity to be too high.
-1058	Cannot create a transition curve in Circle-Line profile with the given transition parameters	The radius of the transition curve is too large. The radius of the transition curve must be smaller than half the circle radius in the next function block. Change the transition parameter to a smaller value.
-1059	Cannot create a transition curve in Line-Circle profile with the given line length	The radius of the transition curve is too large. The radius of the transition curve must be smaller than half the circle radius in the previous function block. Change the transition parameter to a smaller value.
-1060	Cannot create a transition curve in Circle-Line profile with the given line length	G-MAS cannot create a transition curve because the line length in the first function block is less than 1 [user units].
-1061	Cannot create a transition curve in the profile with the given input parameters	G-MAS cannot create a transition curve because the line length in the second function block is less than 1 [user units].

Error	Explanation	Resolution
-1062	Cannot create a transition curve in the profile with the given input parameters	G-MAS cannot create a transition curve because the transition curve does not intersect with the circle. Refer to Elmo for support.
-1063	Cannot create a transition curve in the profile with the given input parameters	G-MAS cannot create transition curve because the transition curve does not intersect with the line. Refer to Elmo for support.
-1064	Cannot create a transition curve with the given transition parameter	G-MAS cannot create transition curve because the transition curve does not intersect with the line. Refer to Elmo for support.
-1065	Cannot create a transition curve with a small radius in the first function block	The transition parameter used for this mode is inappropriate. To create a transition curve between two function blocks using the transition mode MS_TM_SWITCH_RADIUS_MODE, the minimum value of the transition parameter is 1 [user units]
-1066	Cannot create a transition curve	To create a profile with a Circle-Line transition curve, verify that the radius of the first function block (circle motion) Is larger than $1e^{-6}$ [user units]
-1067	Cannot create a Line-Circle transition curve in SwitchRadius mode	Refer to Elmo for support.
-1068	Transition curve cannot be inserted in Line-Line mode - angle between two lines close to 0°	Refer to Elmo for support.
-1069	Not in use	Not in use
-1070	Cannot create a Line-Circle transition curve	The start point of the transition curve does not intersect with the line in the first function block. Refer to Elmo for support.
-1071	Calculation error of the Line-Circle transition curve	Refer to Elmo for support.
-1072	Cannot create Line-Circle transition curve in SwitchRadius mode	Refer to Elmo for support.
-1073	Calculation error in the Circle-Circle transition curve	Refer to Elmo for support.
-1074	Calculation error in the Circle-Line transition curve	Refer to Elmo for support.
-1075	Cannot allocate the spline	Check the format of the Spline input file. If you find a problem in the file, correct it and run the function again.

Error	Explanation	Resolution
		If you cannot locate the problem, try reducing the number of points in the input file.
-1076	Cannot de-allocate the spline	Verify that the current handle is not already de-allocated.
-1077	Error in the Spline input file	Verify that the spline input file is present in the given path. Verify that the format of the file is correct. If the above input file checks are OK, then the error is otherwise. Refer Elmo for support.
-1078	One of the input parameters is wrong	Check that the number of axis members in the group is the same as the spline dimension parameter in the input file. Check whether the spline mode value is allowed. Verify that the axis handle does not exceed the maximum of 20.
-1079	Spline allocation error	If you attempting to move path, first run the MMC_PathSelectCmd function. If you attempting to unselect the path using the MMC_PathUnselectCmd function, perform it once only.
-1080	Spline memory handle is out of range	The spline memory handle should have a value between 1 - 19.
-1081	Cannot create Spline motion with less than 3 points	Create Spline input file with more than 2 points.
-1082	Profiler Error: Auxiliary point to close to Start point	When creating a polynomial segment, always ensure that the distance between Auxiliary and Start point is at least 1E-6. The distance between the points calculated as vector distance between auxiliary and start vector. Distance = $\sqrt{((\mathrm{Aux}[0]-\mathrm{Start}[0])^2+(\mathrm{Aux}[1]-\mathrm{Start}[1])^2 \ldots\ldots(\mathrm{Aux}[n]-\mathrm{Start}[n])^2)}$
-1083	Start point to close to End point	When creating a polynomial segment, always ensure that the distance between Start and End point is at least 1E-6. The distance between the points calculated as vector distance between start and end vector. Distance = $\sqrt{((\mathrm{Start}[0]-\mathrm{End}[0])^2+(\mathrm{Start}[1]-\mathrm{End}[1])^2 \ldots\ldots(\mathrm{Start}[n]-\mathrm{End}[n])^2)}$
-1084	Auxiliary point to close to End point	When creating a polynomial segment, always ensure that the distance between Auxiliary and End point is at least 1E-6. The distance between the points calculated as vector distance between auxiliary and end vector. Distance = $\sqrt{((\mathrm{Aux}[0]-\mathrm{End}[0])^2+(\mathrm{Aux}[1]-\mathrm{End}[1])^2 \ldots\ldots(\mathrm{Aux}[n]-\mathrm{End}[n])^2)}$

Error	Explanation	Resolution
-1085	High ratio between polynomial sub-segments	When creating a polynomial segment, define the three points: * Start point * Auxiliary point * End point These points define two sub-segments * The lenght of the straight line between Auxiliary and Start point - AS. * The lenght of the straight line between Auxiliary and End point - AE. The maximum allowed ratio between these lenght's is "5": * AS/AE lower than 5. * AE/AS lower than 5. Please ensure that the ratio will be within the limits.

6.8. NC Profiler Caution IDs

The following table lists the NC profiler caution IDs. These caution IDs are caused by the G-MAS reading and automatically overcoming an error. These IDs are therefore situations where the G-MAS recalculates the motion profile to overcome a possible issue.

Error	Explanation	Resolution
1001	The transition curve does not enter to Line-Line mode	The given input creates a profile with two coincident lines in the same direction
1002	The transition curve does not enter to Circle-Line mode	The direction in the end-point of the circle is similar to the direction of the line start point.
1003	The transition curve does not enter to Circle-Circle mode	The direction in the end-point of the first circle is similar to the direction of the second circle start-point.
1004	Contradiction between Buffer mode and Transition mode	Set the appropriate Buffer mode for the Transition. If you do not set the appropriate Buffer/Transition parameters, the profiler automatically sets a default parameters: Buffer mode - MC_BUFFERED_MOD Transition mode - MC_TM_NONE_MODE
1005	The function block has set a segment length of zero	This error only relates to G-MAS MultiAxes linear functions, where the G-MAS does not support zero length movement.
1006	Linear Segment length is zero	This error only relates to G-MAS MultiAxes linear functions, where the G-MAS does not support zero length movement. Verify that your destination location is not equal to the start position, or very close to the start location (less than $1e^{-6}$). In a group of axes, one of the axes must move a discrete distance, while the others may remain static. To get the current position, call the function MMC_ReadActualPositionCmd.
1007	Circular Segment length is zero	This error only relates to MultiAxis circular functions, where the G-MAS does not support zero length movement. Verify that your destination location is not equal to the start position, or very close to the start location (less than $1e^{-6}$). In a group of axes, one of the axes must move a discrete distance, while the others may remain static. To get the current position, call the function MMC_ReadActualPositionCmd.

1008	Segment length is zero on ACS mode	This error related only to MultiAxis functions in ACS mode, where the G-MAS does not support zero length movement. Verify that your destination location is not equal to the start position, or very close to the start location (less than $1e^{-6}$). In a group of axes, one of the axes must move a discrete distance, while the others may remain static. To get the current position, call the function MMC_ReadActualPositionCmd.
1009	Single axis Segment length is zero, or, the single axis motion fails	This error only relates to SingleAxis functions, where the G-MAS does not support zero length movement. Verify that your destination location is not equal to the start position, or very close to the start location (less than $1e^{-6}$). This warning is also generated when a single axis motion fails. To get the current position, call the function MMC_ReadActualPositionCmd.

6.9. Internal Library Errors

These are errors, which result in the G-MAS sending a G-MAS library error to the host or other application connected to the G-MAS. Their cause may vary from communication to servo drive errors. The following table lists the Interanl library Errors.

ID	Explanation	Resolution
100	Warning	Get warning from G-MAS or Profiler or GMAS, refer to ErrorID variable to get the exact warning number
0	OK	No error. No warning. All OK.
-1	G-MAS error	Error occurred on GMAS side. Please refer to ErrorID for further information.
-2000	Send data failed.	Refer to Elmo for support.
-2001	Timeout received as response to sending data.	Receive timeout in response from G-MAS, instead of receiving data. Check the communication between the PC and the G-MAS. If the communication is OK, but you still receive this error, refer to Elmo for support.
-2002	Wait response fails	Refer to Elmo for support.
-2003	Received failure response to sending data.	Check the communication between the PC and the G-MAS. If the communication is OK, but you still receive this error, refer to Elmo for support.
-2004	Received response of incorrect data size.	Check the communication between the PC and the G-MAS. If the communication is OK, but you still receive this error, refer to Elmo for support.
-2005	CANbus received response data timeout	Check the cable connections between the G-MAS and the drives. Check the CANbus communication between the G-MAS and the drives. If the communication is OK, but you still receive this error, refer to Elmo for support.
-2006	Function transfer error	The size of the function input structure is corrupt. Make sure that you have inserted a proper input structure for the function. Make sure that the internal variables of the input structure are assigned correctly.

ID	Explanation	Resolution
-2007	Input Data error	When using the Elmo ASCII interpreter, only send supported commands. All commands are case sensitive.
-2008	No UDP Callback connection established.	Before running this function, run the MMC_OpenUdpChannelCmd function. When sending interpreter commands or Send\Receive SDO commands to the Drive you must send the correct axis reference. The axis reference cannot be greater than the number of active axes.
-2009	Unsupported function under the current connection type	You cannot run the MMC_IPCInitConnection function when you working in WIN32 environment. To initialize connection run MMC_RpcInitConnection.

6.10. EtherNetIP Communication Error Codes

The following table describes the EtherNetIP error codes caused by faulty communication.

ID	Error	Resolution
0	EIP_ERR_OK	No Error
-1	EIP_ERR_NO_MEMORY_CREATED	no tag was created on GMAS. Check XML structure
-2	EIP_ERR_REFERENCE_OUT_OF_RANGE	invalide tag reference value.
-3	EIP_ERR_REFERENCE_NOT_FOUND	GMAS cannot find tag by given reference.
-4	EIP_ERR_INVALID_INSTANCE	invalid tag instance value
-5	EIP_ERR_INVALID_BUFFER	Invalid device tag buffer. buffer address is NULL.
-6	EIP_ERR_NETPATH_FAILURE	PLC address is missing. Check device NETWORKPATH in XML
-7	EIP_ERR_INVALID_TAG_NAME	Tag name is missing. Check tag NAME in XML
-8	EIP_ERR_ILLEGAL_UNCONNECTED_REQUEST	General unconnected request error. mainly refers to EthernetIP communication problem with PLC
-9	EIP_ERR_DATA_NOT_RECEIVED	Not in use
-10	EIP_ERR_SET_EVENT_FAILURE	Waiting for device object event failed. Refers to synchronous device tag reading.
-11	EIP_ERR_SESSION_FAIL	EthernetIP session openning failed.
-12	EIP_ERR_EIPTASK_FAIL	EthernetIP task invocation has failed.
-13	EIP_ERR_INIT_MUTEX_FAIL	Insufficient resources. Call ELmo support.
-14	EIP_ERR_NO_OPEN_SESSION	No open session. One may probably has not call to EipOpenSession.
-15	EIP_ERR_MEMORY_CREATED	Memory has already been allocated. EipCreat was already invoced.
-16	EIP_ERR_OPEN_FILE_FAILURE	Failed to open XML file. Check if XML file exists and if path is correct.
-17	EIP_ERR_PARSE_FILE_FAILURE	Failed to parse XML file. Validate XML correctness.
-18	EIP_ERR_CLOSE_FILE_FAILURE	Failed to close XML file. Check write permissions of XML file.

-20	EIP_ERR_ASSEMBLY_CREATE_FAIL	Failed to allocate memory for assemblies. Check definitions in XML.
-21	EIP_ERR_DEVTAG_CREATE_FAIL	Failed to allocate memory for dvice tags. Check definitions in XML.
-22	EIP_ERR_ADPTAG_CREATE_FAIL	Failed to allocate memory for adaptor tags. Check definitions in XML.
-23	EIP_ERR_SEMAPHORE_INIT_FAIL	System error. Call Elmo support.
-24	EIP_ERR_DESTROY_MUTEX_FAIL	System error. Call Elmo support.
-25	EIP_ERR_INVALID_ASSEMBLY	Invalid definition for assembly. Instance must be greater then zero and less then 255. Supported 'COMFORMAT' attribute is: BOOL, SINT, INT, DINT or REAL.
-26	EIP_ERR_INVALID_DEVICE	Invalid definition for device tag. Supported 'TYPE' attribute is: BOOL, SINT, INT, DINT or REAL.
-27	EIP_ERR_INVALID_ADAPTER	Invalid definition for adaptor tag. Supported 'TYPE' attribute is: BOOL, SINT, INT, DINT or REAL.
-28	EIP_ERR_REQUESTS_LIMIT_REACHED	Number of outstanding object requests for device tags exceeded MAX_REQUESTS limit. Call for Elmo support.
-29	EIP_ERR_OUT_OF_MEMORY	Unconnected request out of memory error. Call for Elmo support.
-30	EIP_ERR_INVALID_NETWORK_PATH	Unconnected request invalid network path error. Check device NETWORKPATH in XML

Chapter 7: Error Correction Mechanism

This chapter describes the correction mechanism to correct non-linear mechanical position errors. The correction is performed by calculating pre-defined correction values at discrete position points (as measurement grid points) prior to the motion, creating an error correction table in the G-MAS. In order to compute the error correction value at a given point, a linear interpolation (1D or higher) between two adjacent points in the error correction table is applied, and the value is added to the position calculated by the profiler. This sum is downloaded to the drive and the position correction reported to the user. This is the actual encoder position, which has not changed.

The Position Correction is performed in the low-level code immediately after retrieving the hardware position values, but just before sending the next target position command. This correction is therfore transparent to the end user and considered normal software operation.

In distributed control architecture, the 1D, 2D and 3D error compensations are implemented at the master controller level in the G-MAS, which has overall control of the multiple axes. Therefore, GMAS reads the axes X/Y/Z etc. position data to check the actual position, calculates the Error Compensation Tables, and sends the corrected intermittent target commands via the field bus network.

It is important to note, that when running in NC mode (cyclic position), the correction is performed continuously in real time every Sync cycle time, throughout path execution. In distributed motions, (e.g Profile Position), only the final target position is corrected.

7.1. 2-D Error Correction

2-D Error correction refers to every correction point *(x,y)* on a two dimensional grid is 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. Therefore:

CorrectedPosition=HardwareReading+ErrorCorrection

where, the *ErrorCorrection* is a function of two position inputs. In general, this may be summed as:

$\gamma = f(\alpha,\beta)$, where, γ is the corrected position of any one of the axes. α,β are inputs to γ, and may be any of the G-MAS axes.

In reality, there can be numerous axes in the system (over the field bus). G-MAS support four separate (independant) compensation tables, either in 1D, 2D or 3D with a maximum total of 100000 points allowed for all four tables.

The position inputs for the table are the axes position as reported over the field bus. Usually it is the main servo loop position feedback returning from the drive. The Corrected Position per axis, can then be defined as a mapping function on any two different hardware position inputs. This is demonstrated in **Figure 7-1** overpage.

Figure 7-1: Schematic example of the Corrected Position axis mapping function

Looking at a general α, β grid, and we want to calculate the γ (height) from the grid, as a function of the α, β input axes, it would look similar to the Height defined by the Error Correction Function shown in **Figure** 7-2 below.

Figure 7-2: Error Correction Function 3-D Graph

7.2. 3-D Error Correction

3-D error correction is somewhat similar to 2-D error correction. In 2-D error correction, a 2-D table and grid is defined for any specific two axes. The idea of 3-D error correction is having **multiple layered** grids. As defined for the 2-D mode, the third axis may also be defined as any of the axes, and must be user predefined (as part of the general setup).

Generally, the correction at any point for any specific axis χ, can be defined as a function of any other three axes (α, β, δ):

γ = f (α, β, δ)

For the purpose of a 3-D correction, a set of 2-D corrections, with identical grid points (on the α, β plan) are used. A graphic presentation is shown in **Figure 7-3** below.

Figure 7-3: Error Correction Function layered 3-D Graph

Figure 7-3 displays an m x n x k dimensional grid, where

k = 3 x 2-D grids of 25 x 25 points each. All 2-D grids have the same definitions.

The method involves locating a third point (correction) called δ, defining the third dimension (Z axis in **Figure** 7-3).

Initially, the index of two relevant 2-D m x n grids we are using is to be located. Actually, this requires searching in the δ axis, between the two 2-D grids we are located. We are therefore searching between Layers 1 and 2 (**Figure** 7-3), performing the calculation using the identical 2-D equations on the two grids where we are positioned in the middle. We wish to calculate 2 x γ's (one per 2-D grid) as described in the previous section for the 2-D correction.

7.3. Data Representation

Data representation in the controller has considerable consequences on the code efficiency.

7.3.1. 1-D Representation

The 1D data representation remains intact, and is supported in the G-MAS level. The matrix is represented as a linear matrix.

9	10	11	12

The Error Corrrection file is actually a tab-separated text file, which can be created in Excel and takes the following format (example), for say each α and β value above.

1D Example

Table size	5
Start offset	0
Error table dimension	1
Start position	20000
Axis grid size	16
Table dimensions	5

The final data table appears as shown below:

Table #1 Start Data

0	100	0	100	0

Table #1 End Data

7.3.2. 2-D Representation

The 2-D data representation involves the error correction data for 2-D grid points, stored in the same ET. The matrix is represented as a linear matrix.

9	10	11	12
5	6	7	8
1	2	3	4

In the above matrix, the starting ET for the α axis is 1 (but not necessarily). The start ET for the β axis is ETStartIndex + α size. The size is the m x n size of the grid. Each number in the above matrix represents the value of the error correction in the given (α, β) point. The start index in the memory is not 1, but is provided by the user.

In the 2-D case, the user must enter the following additional data (besides the correction data):

1. For each axis, the user must define the input axes α, β (which axis are α, β from the possible G-MAS Axis nodes).
2. The Error Corrrection file is actually a tab-separated text file which can be created in Excel, and takes the following format (example), for say each α and β value above.

2D **Example**

Table size	25		This is the total table size (actual number of points)
Start offset	0		The offset in the shared memory segment where the table will start. If the physical address is not available – this function will fail
Error table dimension	2		Either 1D/2D/3D
Start position	20000	20000	The position the error correction should start (only for α as the start point for β is last point of α)
Axis grid size	16	16	The actual resolution between sample points. Given as a power of 2. For example, in this table the grid size is 16, therefore the interval between 2 adjacent points is 216 = 65536.
Table dimensions	5	5	Number of rows and columns. In a 3D table, the third parameter will be the number of tables.

The final data table appears as shown below:

Table #1 Start Data

0	100	0	100	0
0	100	0	100	0
0	100	0	100	0
0	100	0	100	0
0	100	0	100	0

Table #1 End Data

7.3.3. 3-D Representation

The 3-D data representation the data stored in the same data AR. The 3-D data is saved as consecutive 2-D linear matrices.

9	10	11	12
5	6	7	8
1	2	3	4

Grid#1

21	22	23	24
17	18	19	20
13	14	15	16

Grid#2

etc… for additional grid's.

1. For each axis, the user must define the input axes α, β, δ (of which axis are α, β, δ, from the possible G-MAS Axis nodes).
2. For each α, β, and δ value, the following must be inserted (refer to the 2-D Example **on page 7-470 (2D Example)**):
   a. Table size
   b. Start offset
   c. Error table dimension
   d. Start position (only for α as the start point for β is last point of α, δ is the last point of β)
   e. Axis grid size
   f. Table dimensions

7.4. Error Correction Functions

The following Error Correction functions are described:

Error Correction functions
MMC_LoadErrorCorrTable
MMC_UnloadErrorCorrTable
MMC_EnableErrorCorrTable
MMC_DisableErrorCorrTable
MMC_GetErrorTableStatus

7.4.1. MMC_LoadErrorCorrTable

Loads an error correction table to memory. Error correction is then performed according to this table.

```
MMC_LIB_API int MMC_LoadErrorCorrTableCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_LOADERRORTABLE_IN* pInParam,
OUT MMC_LOADERRORTABLE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS/includes/MMC_ErrorCorr_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_LOADERRORTABLE_IN** input data structure using the MMC_LoadErrorCorrTable function.

pOutParam

Points to the **MMC_LOADERRORTABLE_OUT** output structure receiving information, as a result of calling the MMC_LoadErrorCorrTable function.

Remarks

The Error Correction Table is loaded to the memory and requires enabling using the function **MMC_EnableErrorCorrTable**, in order to be functional. If loaded, and enabled, the table can be retained in memory but disabled using the function **MMC_DisableErrorCorrTable**. To change its functionality to enable, perform MMC_EnableErrorCorrTable again. To unload the table from the memory perform the function **MMC_UnloadErrorCorrTable**.

Scope

All

MMC_LOADERRORTABLE_IN Structure

```
typedef struct{
NC_NODE_HNDL_T hMappedAxisRef[NC_ERROR_TABLE_DIMENSION_3D];
NC_NODE_HNDL_T hTargetAxisRef;
double dMaxCorrectionDelta;
NC_ERROR_TABLE_NUMBER eETNumber;
unsigned char pPathToETFile[NC_MAX_ET_FILE_PATH_LENGTH];
}MMC_LOADERRORTABLE_IN;
```

Parameters

NC_NODE_HNDL_T hMappedAxisRef

The error correction is performed according to these axes. The NC_NODE_HNDL_T is an enumerator that defines the Node handle. *hMappedAxisRef* defines the axes involved in the linear interpolation procedure.

[NC_ERROR_TABLE_DIMENSION] defines the dimensions of the error table, and is an enumerator with values:

NC_ERROR_TABLE_DIMENSION_NONE = 0

NC_ERROR_TABLE_DIMENSION_1D = 1

NC_ERROR_TABLE_DIMENSION_2D = 2

NC_ERROR_TABLE_DIMENSION_3D = 3

NC_ERROR_TABLE_DIMENSION_MAX = 4

The default is NC_ERROR_TABLE_DIMENSION_3D = 3 which can be applied to 1D – 3D tables inclusive.

NC_NODE_HNDL_T hTargetAxisRef

The parameter *hTargetAxisRef* defines the Axis reference of target axis, dependant on the handler.

The NC_NODE_HNDL_T is the enumerator handler that defines the Node handle transition.

dMaxCorrectionDelta

Parameter currently not in use

eETNumber

Defines the error table letter assigned.

NC_ERROR_TABLE_NUMBER is an enumerator describing the with the following values:

NC_ERROR_TABLE_A

NC_ERROR_TABLE_B

NC_ERROR_TABLE_C

NC_ERROR_TABLE_D

NC_ERROR_TABLE_MAX

pPathToETFile

Defines the path to the error table file.

[NC_MAX_ET_FILE_PATH_LENGTH] is the maximum size of the error table file path. It is limited to 100.

MMC_LOADERRORTABLE_OUT Structure

typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_LOADERRORTABLE_OUT;

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within the function. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Displays an error code as -ve or +ve integers.

Figure 7-4 describes the function for MMC_LoadErrorCorrTable

Figure 7-4: MMC_LoadErrorCorrTable function

7.4.1.2. Function Code Example

```
MMC_LOADERRORTABLE_IN       stLoadErrorTableIn;
MMC_LOADERRORTABLE_OUT      stLoadErrorTableOut;

stLoadErrorTableIn.eETNumber              = NC_ERROR_TABLE_A;
stLoadErrorTableIn.hMappedAxisRef[0]      = hAxisRef1;
stLoadErrorTableIn.hMappedAxisRef[1]      = hAxisRef2;
stLoadErrorTableIn.hTargetAxisRef         = hAxisRef3;
strcpy(stLoadErrorTableIn.pPathToETFile, szFileName);

rc = MMC_LoadErrorCorrTableCmd(hConnHndl, &stLoadErrorTableIn, &stLoadErrorTableOut);
if (NC_OK != rc)
{
HandleError();
}
```

7.4.2. MMC_EnableErrorCorrTable

Enables the usage of an error table.

```
MMC_LIB_API int MMC_EnableErrorCorrTableCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_ENABLEERRORTABLE_IN* pInParam,
OUT MMC_ENABLEERRORTABLE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS/includes/MMC_ErrorCorr_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_ENABLEERRORTABLE_IN** input data structure using the MMC_EnableErrorCorrTable function.

pOutParam

Points to the **MMC_ENABLEERRORTABLE_OUT** output structure receiving information, as a result of calling the MMC_EnableErrorCorrTable function.

Remarks

The Error Correction Table is loaded to the memory and requires enabling using the function **MMC_EnableErrorCorrTable**, in order to be functional. If loaded, and enabled, the table can be retained in memory but disabled using the function **MMC_DisableErrorCorrTable**. To change its functionality to enable, perform MMC_EnableErrorCorrTable again. To unload the table from the memory perform the function **MMC_UnloadErrorCorrTable**.

Scope

All

MMC_ENABLEERRORTABLE_IN Structure

```
typedef struct{
NC_ERROR_TABLE_NUMBER eTableNumber;
}MMC_ENABLEERRORTABLE_IN;
```

Parameters

eTableNumber

Defines the error table letter assigned to be enabled.

NC_ERROR_TABLE_NUMBER is an enumerator describing the with the following values:

NC_ERROR_TABLE_A

NC_ERROR_TABLE_B

NC_ERROR_TABLE_C

NC_ERROR_TABLE_D

NC_ERROR_TABLE_MAX

MMC_ENABLEERRORTABLE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_ENABLEERRORTABLE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within the function. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 7-5 describes the function for MMC_EnableErrorCorrTable

Figure 7-5: MMC_EnableErrorCorrTable function

7.4.2.2. Function Code Example

```
MMC_ENABLEERRORTABLE_IN stEnableErrorTableIn;
MMC_ENABLEERRORTABLE_OUT stEnableErrorTableOut;

stEnableErrorTableIn.eTableNumber = NC_ERROR_TABLE_A;

rc = MMC_EnableErrorCorrTableCmd(hConnHndl, &stEnableErrorTableIn, &stEnableErrorTableOut);
if (NC_OK != rc)
{
HandleError();
}
```

7.4.3. MMC_GetErrorTableStatus

Function recieves the table number as input and returns an answer whether the table is loaded and/or enabled.

```
MMC_LIB_API int MMC_GetErrorTableStatusCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETERRORTABLESTATUS_IN* pInParam,
OUT MMC_GETERRORTABLESTATUS_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS/includes/MMC_ErrorCorr_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETERRORTABLESTATUS_IN** input data structure using the MMC_GetErrorTableStatus function.

pOutParam

Points to the **MMC_GETERRORTABLESTATUS_OUT** output structure receiving information, as a result of calling the MMC_GetErrorTableStatus function.

Remarks

The Error Correction Table is loaded to the memory and requires enabling using the function **MMC_EnableErrorCorrTable**, in order to be functional. If loaded, and enabled, the table can be retained in memory but disabled using the function **MMC_DisableErrorCorrTable**. To change its functionality to enable, perform MMC_EnableErrorCorrTable again. To unload the table from the memory perform the function **MMC_UnloadErrorCorrTable**.

Scope

All

MMC_GETERRORTABLESTATUS_IN Structure

```
typedef struct{
NC_ERROR_TABLE_NUMBER eTableNumber;
}MMC_GETERRORTABLESTATUS_IN;
```

Parameters

eTableNumber

Defines the error table letter assigned.

NC_ERROR_TABLE_NUMBER is an enumerator describing the with the following values:

NC_ERROR_TABLE_A

NC_ERROR_TABLE_B

NC_ERROR_TABLE_C

NC_ERROR_TABLE_D

NC_ERROR_TABLE_MAX

MMC_GETERRORTABLESTATUS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned char ucIsTableEnabled;
unsigned char ucIsTableLoaded;
}MMC_GETERRORTABLESTATUS_OUT;
```

Parameters

ucIsTableEnabled

Returns the Boolean solution to the question, whether the table is **enabled** or not.

ucIsTableLoaded

Returns the Boolean solution to the question, whether the table is **loaded** or not.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within the function. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 7-6 describes the function for MMC_GetErrorTableStatus

Figure 7-6: MMC_GetErrorTableStatus function

7.4.3.2. Function Code Example

```
MMC_GETERRORTABLESTATUS_IN stGetErrorTableStatusIn;
MMC_GETERRORTABLESTATUS_OUT stGetErrorTableStatusOut;

stGetErrorTableStatusIn.eTableNumber = NC_ERROR_TABLE_A;

rc = MMC_GetErrorTableStatusCmd(hConnHndl, &stGetErrorTableStatusIn,
&stGetErrorTableStatusOut);

if (NC_OK != rc)
{
HandleError();
}
```

7.4.4. MMC_DisableErrorCorrTable

Disables the usage of an error table.

```
MMC_LIB_API int MMC_DisableErrorCorrTableCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_DISABLEERRORTABLE_IN* pInParam,
OUT MMC_DISABLEERRORTABLE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS/includes/MMC_ErrorCorr_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_DISABLEERRORTABLE_IN** input data structure using the MMC_DisableErrorCorrTable function.

pOutParam

Points to the **MMC_DISABLEERRORTABLE_OUT** output structure receiving information, as a result of calling the MMC_DisableErrorCorrTable function.

Remarks

The Error Correction Table is loaded to the memory and requires enabling using the function **MMC_EnableErrorCorrTable**, in order to be functional. If loaded, and enabled, the table can be retained in memory but disabled using the function **MMC_DisableErrorCorrTable**. To change its functionality to enable, perform MMC_EnableErrorCorrTable again. To unload the table from the memory perform the function **MMC_UnloadErrorCorrTable**.

Scope

All

MMC_DISABLEERRORTABLE_IN Structure

```
typedef struct{
NC_ERROR_TABLE_NUMBER eTableNumber;
}MMC_DISABLEERRORTABLE_IN;
```

Parameters

eTableNumber

Defines the error table letter assigned to be disabled.

NC_ERROR_TABLE_NUMBER is an enumerator describing the with the following values:

NC_ERROR_TABLE_A

NC_ERROR_TABLE_B

NC_ERROR_TABLE_C

NC_ERROR_TABLE_D

NC_ERROR_TABLE_MAX

MMC_DISABLEERRORTABLE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_DISABLEERRORTABLE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within the function. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 7-7 describes the function for MMC_DisableErrorCorrTable

Figure 7-7: MMC_DisableErrorCorrTable function

7.4.4.2. Function Code Example

```
MMC_DISABLEERRORTABLE_IN stDisableErrorTableIn;
MMC_DISABLEERRORTABLE_OUT stDisableErrorTableOut;

stDisableErrorTableIn.eTableNumber = NC_ERROR_TABLE_A;

rc = MMC_DisableErrorCorrTableCmd(hConnHndl, &stDisableErrorTableIn, &stDisableErrorTableOut);
if (NC_OK != rc)
{
HandleError();
}
```

7.4.5. MMC_UnloadErrorCorrTable

Unloads an error correction table from memory.

```
MMC_LIB_API int MMC_UnloadErrorCorrTableCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_UNLOADERRORTABLE_IN* pInParam,
OUT MMC_UNLOADERRORTABLE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS/includes/MMC_ErrorCorr_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_UNLOADERRORTABLE_IN** input data structure using the MMC_UnloadErrorCorrTable function.

pOutParam

Points to the **MMC_UNLOADERRORTABLE_OUT** output structure receiving information, as a result of calling the MMC_UnloadErrorCorrTable function.

Remarks

The Error Correction Table is loaded to the memory and requires enabling using the function **MMC_EnableErrorCorrTable**, in order to be functional. If loaded, and enabled, the table can be retained in memory but disabled using the function **MMC_DisableErrorCorrTable**. To change its functionality to enable, perform MMC_EnableErrorCorrTable again. To unload the table from the memory perform the function **MMC_UnloadErrorCorrTable**.

Scope

All

MMC_UNLOADERRORTABLE_IN Structure

```
typedef struct{
NC_ERROR_TABLE_NUMBER eTableNumber;
}MMC_UNLOADERRORTABLE_IN;
```

Parameters

eTableNumber

Defines the error table letter assigned to be unloaded.

NC_ERROR_TABLE_NUMBER is an enumerator describing the with the following values:

NC_ERROR_TABLE_A

NC_ERROR_TABLE_B

NC_ERROR_TABLE_C

NC_ERROR_TABLE_D

NC_ERROR_TABLE_MAX

MMC_UNLOADERRORTABLE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_UNLOADERRORTABLE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within the function. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 10-29 describes the function for MMC_UnloadErrorCorrTable

Figure 7-8: MMC_UnloadErrorCorrTable function

7.4.5.2. Function Code Example

```
MMC_UNLOADERRORTABLE_IN     stUnloadErrorTableIn;
MMC_UNLOADERRORTABLE_OUT    stUnloadErrorTableOut;

stUnloadErrorTableIn.eTableNumber = NC_ERROR_TABLE_A;

rc = MMC_UnloadErrorCorrTableCmd(hConnHndl, &stUnloadErrorTableIn, &stUnloadErrorTableOut);

if (NC_OK != rc)
{
HandleError();
}
```

Chapter 8: Bulk Parameters Reading

This set of functions allows the user to retrieve all the parameters for a limited number of single axes operating simultaneously using a single function call. The G-MAS allows a user program or host via Ethernet interface to retrieve all required axis parameters. As the number of axes and parameters increases, the time to retrieve the parameters increases proportionately. This operation is relatively slow, and this chapter explains the purpose of bulk read, which increases the performance of the parameters retrieval procedure dramatically.

This feature can be used in two difference scenarios:

- To import multiple parameters when using G-MAS Multiple Axes in the EAS software
- To retrieve multiple parameters for multiple single axis when programming in C or C++ (or any other host software)

8.1. Bulk Reading Functions

The following Bulk Reading functions are described:

Bulk Reading functions
MMC_ConfigBulkRead
MMC_PerformBulkRead

8.1.1. MMC_ConfigBulkRead

Configures the function to read all paramaters from multiple axes.

```
MMC_LIB_API int MMC_ConfigBulkReadCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_CONFIGBULKREAD_IN* pInParam,
OUT MMC_CONFIGBULKREAD_OUT* pOutParam
);
```

Motion Mode	NC – Immaterial	Distributed – Immaterial
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_CONFIGBULKREAD_IN** input data structure using the MMC_ConfigBulkRead function.

pOutParam

Points to the **MMC_CONFIGBULKREAD_OUT** output structure receiving information, as a result of calling the MMC_ConfigBulkRead function.

Remarks

The G-MAS has three function block queues,large, medium, and small, function blocks. These are common to all nodes. Every time a function block is inserted, a function block is taken from one of these queues. The parameters *iFreeLargeFbsNumber, iFreeMediumFbsNumber and iFreeSmallFbsNumber* indicate the number of free function blocks available in each queue.

Scope

All

MMC_CONFIGBULKREAD_IN Structure

```
typedef struct mmc_configbulkread_in{
NC_BULKREAD_PARAMETERS_UNION uBulkReadParams;
NC_BULKREAD_CONFIG_ENUM eConfiguration;
unsigned short usAxisRefArray[NC_MAX_AXES_PER_BULK_READ];
unsigned short usNumberOfAxes;
unsigned char ucIsPreset;
}MMC_CONFIGBULKREAD_IN;
```

Parameters

uBulkReadParams

Defines what parameters will be read. It can be either one of the predefined presets, or a custom array with user filled values. ulBulkReadParameters is an array of size 32 Bit, and to set the number of signals to be read from the bulk read user, it is necessary to set the value of theis parameter to -1 after the signals requested. If two signals are required, then the ulBulkReadParameters[2] value should be -1 otherwise a junk value that was in the array could be consider as a signal requsted

The Union parameter NC_BULKREAD_PARAMETERS_UNION is defined as follows:

```
typedef union{
NC_BULKREAD_PRESET_ENUM eBulkReadPreset;
unsigned long ulBulkReadParameters[NC_MAX_REC_SIGNALS_NUM];
} NC_BULKREAD_PARAMETERS_UNION;
```

Where the user can select to use either the parameters:

NC_BULKREAD_PRESET_ENUM eBulkReadPreset

Or

unsigned long ulBulkReadParameters[NC_MAX_REC_SIGNALS_NUM]

eBulkReadPreset

The bulk read preset enumerator defined by NC_BULKREAD_PRESET_ENUM with values:

```
eNC_BULKREAD_PRESET_NONE,
eNC_BULKREAD_PRESET_1,
eNC_BULKREAD_PRESET_2,
eNC_BULKREAD_PRESET_3,
eNC_BULKREAD_PRESET_4,
eNC_BULKREAD_PRESET_5,
eNC_BULKREAD_PRESET_MAX,
```

Each of the bulk read presets defines a fixed set of parameters to be read:

```
typedef struct eNC_BULKREAD_PRESET_1{
int aPos;
int aVel;
int aTorque;
unsigned long ulAxisStatus;
```

```
unsigned int uiInputs;
OPM402 eOpMode;
}NC_BULKREAD_PRESET_1;

typedef struct eNC_BULKREAD_PRESET_2{
NC_BULKREAD_PRESET_1 stAxisParams;
int iFreeLargeFbsNumber;
int iFreeMediumFbsNumber
int iFreeSmallFbsNumber;
}NC_BULKREAD_PRESET_2;

typedef struct eNC_BULKREAD_PRESET_3{
int aPos;
int aVel;
int aTorque;
unsigned long ulAxisStatus;
unsigned int uiInputs;
OPM402 eOpMode;
NC_BULKREAD_VARIABLE_UNION ucCommError;
NC_BULKREAD_VARIABLE_UNION usLastEmcyErrorCode;
NC_BULKREAD_VARIABLE_UNION usControlWord;
NC_BULKREAD_VARIABLE_UNION usStatusWord;
}NC_BULKREAD_PRESET_3;

typedef struct eNC_BULKREAD_PRESET_4{
int aPos;
int aHWPos;
int iPosFollowingErr;
int aVel;
int aTorque;
unsigned long ulAxisStatus;
unsigned int uiInputs;
OPM402 eOpMode;
unsigned int uiStatusRegister;
unsigned int uiMcsLimitRegister;
}NC_BULKREAD_PRESET_4;

typedef struct eNC_BULKREAD_PRESET_5{
int aPos;
int aHWPos;
int iPosFollowingErr;
int aVel;
int aTorque;
unsigned long ulAxisStatus;
unsigned int uiInputs;
OPM402 eOpMode;
unsigned int uiStatusRegister;
unsigned int uiMcsLimitRegister;
NC_BULKREAD_VARIABLE_UNION usLastEmcyErrorCode;
NC_BULKREAD_VARIABLE_UNION usControlWord;
NC_BULKREAD_VARIABLE_UNION usStatusWord;
```

```
NC_BULKREAD_VARIABLE_UNION ucCommError;
}NC_BULKREAD_PRESET_5;

typedef union{
char cVar;
unsigned char ucVar;
short sVar;
unsigned short usVar;
int iVar;
unsigned int uiVar;
long lVar;
unsigned long ulVar;
}NC_BULKREAD_VARIABLE_UNION;
```

Where the user can select to use either of the parameters:

Character cVar, Unsigned character ucVar, short sVar, unsigned short usVar, integer iVar, insigned integer uiVar, long lVar, or un signed long ulVar.

Var is the value of the variable.

Each of these preset parameters have the following definitions:

aPos

Actual position integer. Integer values

aHWPos

Actual hardware position. Integer values

iPosFollowingErr

Position following error. Integer values/

aVel

Actual velocity integer

aTorque

Actual Torque integer

ulAxisStatus

Status of the axis with +ve bitwise values

uiInputs

Digital inputs with any +ve integer value

eOpMode

Motion mode

uiStatusRegister

Variable provides information on the special status of an axis. Refer to the chapter **3.2 Interfaces** . +ve integer value.

uiMcsLimitRegister

Parameter represents the status of the MCS limits in a specific group. +ve integer value.

iFreeLargeFbsNumber

Number of large free function blocks avaliable in the queue for this size.

iFreeMediumFbsNumber

Number of medium sized free function blocks avaliable in the queue for this size. Integer value accepted

iFreeSmallFbsNumber

Number of small free function blocks avaliable in each queue. Integer value accepted

usControlWord

CANopen DS402 control word. Any +ve short value.

usStatusWord

CANopen DS402 status word. Any +ve short value.

ucCommError

Input axis communication error. +ve character values.

usLastEmcyErrorCode

Last recorded emergency code. usLastEmcyErrorCode is the emergency error code received from the drive.

ulBulkReadParameters

The array of ulBulkReadParameters defined by [NC_MAX_REC_SIGNALS_NUM] represents a set of parameters to be retrieved from the G-MAS, with a maximum of 32 parameters.

eConfiguration

Defines the reading source. eBULKREAD_CONFIG_1 is reserved to the EAS application. Acceptable values are eBULKREAD_CONFIG_1 and eBULKREAD_CONFIG_2.

NC_BULKREAD_CONFIG_ENUM defines the following values:

eBULKREAD_CONFIG_NONE,

eBULKREAD_CONFIG_1,

eBULKREAD_CONFIG_2,

eBULKREAD_CONFIG_MAX,

usAxisRefArray

Defines the array that will contain the axisrefs to be read (not masked), where [NC_MAX_AXES_PER_BULK_READ] has a range between 1 and 100.

If an error is created, it should return the NC_BULK_READ_NUM_OF_AXES_OUT_OF_RANGE error.

usNumberOfAxes

Defines the number of axes, and is the total number of axis to be bulk read.

ucIsPreset

Whether the preset parameters is used or not. Values accepted are 0, or 1.

MMC_CONFIGBULKREAD_OUT Structure

```
typedef struct mmc_configbulkread_out{
float fFactorsArray[NC_MAX_BULK_READ_READABLE_PACKET_SIZE];
unsigned short usStatus;
short usErrorID;
} MMC_CONFIGBULKREAD_OUT;
```

Parameters

fFactorsArray

Defines what multiplication factor is needed to apply to each read parameter. Dependant on the array [NC_MAX_BULK_READ_READABLE_PACKET_SIZE] with values of 0 – 350.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within the function. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 8-1 describes the function for MMC_ConfigBulkRead.

Figure 8-1: MMC_ConfigBulkRead function

8.1.1.2. Function Code Example

```
MMC_CONFIGBULKREAD_IN stCfgIn;
MMC_CONFIGBULKREAD_OUT stCfgOut;

int rc = NC_OK;

stCfgIn.eConfiguration                  = eBULKREAD_CONFIG_2;
stCfgIn.uBulkReadParams.eBulkReadPreset = eNC_BULKREAD_PRESET_2;
stCfgIn.ucIsPreset                      = 1;
stCfgIn.usAxisRefArray[0]               = 0;
stCfgIn.usAxisRefArray[1]               = 1;
stCfgIn.usNumberOfAxes                  = 2;

rc = MMC_ConfigBulkReadCmd(g_hConnectHndl, &stCfgIn, &stCfgOut);

etc.
```

8.1.1.3. Implementation Example

8.1.2. MMC_PerformBulkRead

Reads those parameters which were configured by a call to ConfigBulkRead, from multiple axes.

```
MMC_LIB_API int MMC_PerformBulkReadCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_PERFORMBULKREAD_IN* pInParam,
OUT MMC_PERFORMBULKREAD_OUT* pOutParam
);
```

Motion Mode	NC – Immaterial	Distributed – Immaterial
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_PERFORMBULKREAD_IN** input data structure using the MMC_PerformBulkRead function.

pOutParam

Points to the **MMC_PERFORMBULKREAD_OUT** output structure receiving information, as a result of calling the MMC_PerformBulkRead function.

Remarks

None

Scope

All

MMC_PERFORMBULKREAD_IN Structure

```
typedef struct mmc_performbulkread_in{
NC_BULKREAD_CONFIG_ENUM eConfiguration;
} MMC_PERFORMBULKREAD_IN;
```

Parameters

eConfiguration

Defines the reading source. eBULKREAD_CONFIG_1 is reserved to the EAS application. Acceptable values are eBULKREAD_CONFIG_1 and eBULKREAD_CONFIG_2.

NC_BULKREAD_CONFIG_ENUM defines the following values:

eBULKREAD_CONFIG_NONE,

eBULKREAD_CONFIG_1,

eBULKREAD_CONFIG_2,

eBULKREAD_CONFIG_MAX,

MMC_PERFORMBULKREAD_OUT Structure

```
typedef struct mmc_performbulkread_out{
unsigned long ulOutBuf[NC_MAX_BULK_READ_READABLE_PACKET_SIZE];
NC_BULKREAD_PRESET_ENUM eChosenPreset;
unsigned short usStatus;
short usErrorID;
} MMC_PERFORMBULKREAD_OUT;
```

Parameters

ulOutBuf

Defines the output buffer read data with a maximum [NC_MAX_BULK_READ_READABLE_PACKET_SIZE] array size of 350.

eChosenPreset

The bulk read preset enumerator defined by NC_BULKREAD_PRESET_ENUM with values:

eNC_BULKREAD_PRESET_NONE,
eNC_BULKREAD_PRESET_1,
eNC_BULKREAD_PRESET_2,
eNC_BULKREAD_PRESET_3,
eNC_BULKREAD_PRESET_4,
eNC_BULKREAD_PRESET_5,
eNC_BULKREAD_PRESET_MAX,

If a user defined parameters array is used for ulBulkReadParameters[NC_MAX_REC_SIGNALS_NUM], then the value of *eChosenPreset* will be eNC_BULKREAD_PRESET_MAX.

Each of the bulk read presets defines a fixed set of parameters to be read:

```
typedef struct eNC_BULKREAD_PRESET_1{
int aPos;
int aVel;
int aTorque;
unsigned long ulAxisStatus;
unsigned int uiInputs;
OPM402 eOpMode;
}NC_BULKREAD_PRESET_1;

typedef struct eNC_BULKREAD_PRESET_2{
NC_BULKREAD_PRESET_1 stAxisParams;
int iFreeLargeFbsNumber;
int iFreeMediumFbsNumber
int iFreeSmallFbsNumber;
}NC_BULKREAD_PRESET_2;

typedef struct eNC_BULKREAD_PRESET_3{
int aPos;
int aVel;
int aTorque;
unsigned long ulAxisStatus;
unsigned int uiInputs;
OPM402 eOpMode;
NC_BULKREAD_VARIABLE_UNION ucCommError;
NC_BULKREAD_VARIABLE_UNION usLastEmcyErrorCode;
NC_BULKREAD_VARIABLE_UNION usControlWord;
NC_BULKREAD_VARIABLE_UNION usStatusWord;
}NC_BULKREAD_PRESET_3;

typedef struct eNC_BULKREAD_PRESET_4{
int aPos;
int aHWPos;
int iPosFollowingErr;
int aVel;
int aTorque;
unsigned long ulAxisStatus;
unsigned int uiInputs;
OPM402 eOpMode;
unsigned int uiStatusRegister;
unsigned int uiMcsLimitRegister;
}NC_BULKREAD_PRESET_4;

typedef struct eNC_BULKREAD_PRESET_5{
int aPos;
int aHWPos;
int iPosFollowingErr;
int aVel;
int aTorque;
unsigned long ulAxisStatus;
```

```
unsigned int uiInputs;
OPM402 eOpMode;
unsigned int uiStatusRegister;
unsigned int uiMcsLimitRegister;
NC_BULKREAD_VARIABLE_UNION usLastEmcyErrorCode;
NC_BULKREAD_VARIABLE_UNION usControlWord;
NC_BULKREAD_VARIABLE_UNION usStatusWord;
NC_BULKREAD_VARIABLE_UNION ucCommError;
}NC_BULKREAD_PRESET_5;
```

```
typedef union{
char cVar;
unsigned char ucVar;
short sVar;
unsigned short usVar;
int iVar;
unsigned int uiVar;
long lVar;
unsigned long ulVar;
}NC_BULKREAD_VARIABLE_UNION;
```

Where the user can select to use either of the parameters:

Character cVar, Unsigned character ucVar, short sVar, unsigned short usVar, integer iVar, insigned integer uiVar, long lVar, or un signed long ulVar.

Var is the value of the variable.

Each of these preset parameters have the following definitions:

aPos

Actual position integer. Integer values

aHWPos

Actual hardware position. Integer values

iPosFollowingErr

Position following error. Integer values/

aVel

Actual velocity integer

aTorque

Actual Torque integer

ulAxisStatus

Status of the axis with +ve bitwise values

uiInputs

Digital inputs with any +ve integer value

eOpMode

Motion mode

uiStatusRegister

Variable provides information on the special status of an axis. Refer to the chapter **3.2 Interfaces**. +ve integer value.

uiMcsLimitRegister

Parameter represents the status of the MCS limits in a specific group. +ve integer value.

iFreeLargeFbsNumber

Number of large free function blocks avaliable in the queue for this size.

iFreeMediumFbsNumber

Number of medium sized free function blocks avaliable in the queue for this size. Integer value accepted

iFreeSmallFbsNumber

Number of small free function blocks avaliable in each queue. Integer value accepted

usControlWord

CANopen DS402 control word. Any +ve short value.

usStatusWord

CANopen DS402 status word. Any +ve short value.

ucCommError

Input axis communication error. +ve character values.

usLastEmcyErrorCode

Last recorded emergency code. usLastEmcyErrorCode is the emergency error code received from the drive.

ulBulkReadParameters

The array of ulBulkReadParameters defined by [NC_MAX_REC_SIGNALS_NUM] represents a set of parameters to be retrieved from the G-MAS, with a maximum of 32 parameters.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within the function. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 8-2 describes the function for MMC_PerformBulkRead.

Figure 8-2: MMC_PerformBulkRead function

8.1.2.2. Function Code Example

```
MMC_PERFORMBULKREAD_IN stPerformBulkReadIn;
MMC_PERFORMBULKREAD_OUT stPerformBulkReadOut;
int rc = NC_OK;

stPerformBulkReadIn.eConfiguration = eBULKREAD_CONFIG_2;
rc = MMC_PerformBulkReadCmd(g_hConnectHndl, &stPerformBulkReadIn, &stPerformBulkReadOut);
if (NC_OK != rc)
{
HandleError();
}
```

8.1.2.3. Implementation Example

Chapter 9: Position, Velocity, Time (PVT) Motion

9.1. Overview

The PV/PVT special motion class 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 PV motion is stored as a table in the G-MAS RAM using a function. The PV motion can operate in "normal" or cyclic modes. In cyclic mode, the profiler reaches the end of table and returns to the beginning.

9.2. PV, PVT Profiler

Generally, PV/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. The path is calculated on-the-fly, and therefore all polynomial coefficient calculations occur in the real-time module. The PV/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 PV/PVT function blocks are only applicable in NC cyclic/interpolated modes.

9.3. Data loading

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:

$$T_n \quad P_{n1} \; V_{n1} \; P_{n2} \; V_{n2} \; \ldots \; P_{nm} \; V_{nm}$$

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.

Note: **Appending data to a table loaded from file is forbidden.**
If T equals the cycle time, a simpler profiler code should be applied.

9.4. PVT motion

The motion is performed using a special type of function block. When this function block type is inserted, a special profiler is applied that calculates polynomial coefficients and moves along the calculated trajectory.

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).

9.5. 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.

9.5.1. Initializing 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.

9.5.2. Loading Data

There are two modes of load data from array – Static and Dynamic. Static mode is pretty similar to load from file mode, the table is loaded and no appending is allowed, when the table is filled with maximum points.

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 to the user space. Using the existing events mechanism – the user handles the overflow as necessary.

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 algorithim 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.

9.5.3. 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

9.6. File example

The points are read as doubles. In practice each row will take

((M*2 + 1) * sizeof(double)) bytes where the 1 stands for time column.

The total memory "allocated" will be (N * (M*2 + 1) * sizeof(double)).

```
PV mode:     X
Dimension:   M
Number of points: N
Time Absolute:  1/0
Position Absolute: 1/0
Cyclic: 1/0

### Data start ###
T1   P11 V11 P12 V12 … P1m V1m
T2   P21 V21 P22 V22 … P2m V2m
.
..
Tn   Pn1 Vn1 Pn2 Vn2 … Pnm Vnm
### Data end ###
```

9.7. PVT Functions

The following PVT functions are described:

PVT functions
MMC_InitTable
MMC_LoadTableFromFile
MMC_AppendPointsToTable
MMC_UnloadTable
MMC_MoveTable
MMC_GetTableIndex

9.7.1. MMC_InitTable

This function allocates a memory segment in the shared memory according to the dimension and number of points.

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

Motion Mode	NC – Supported	Distributed – Not supported
Source	GMAS\includes\MMC_general_API.h GMAS\includes\MMC_PVT_ECAM_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_INITTABLE_IN** input data structure using the MMC_InitTableCmd.

pOutParam

Points to the **MMC_INITTABLE_OUT** output structure receiving information, as a result of calling the MMC_InitTableCmd.

Remarks

It should be noted that when using this function, no data is loaded. The function returns hMemHandle, which is the unique path ID (similar to splines).

Scope

All

MMC_INITTABLE_IN Structure

```
typedef struct mmc_inittable_in{
NC_MOTION_TABLE_TYPE_ENUM eTableType;
unsigned long ulMaxNumberOfPoints;
unsigned long ulUnderflowThreshold;
unsigned short usAxisRef;
unsigned short usDimension;
unsigned char ucIsDynamicMode;
unsigned char ucIsPosAbsolute;
unsigned char ucIsCyclic;
} MMC_INITTABLE_IN;
```

Parameters

eTableType

The Table type defined according to the enumerator parameter NC_MOTION_TABLE_TYPE_ENUM defined by:

eNC_TABLE_NONE = 0

eNC_TABLE_SPLINE = 1

eNC_TABLE_PVT_FILE = 2

eNC_TABLE_PVT_ARRAY = 3

eNC_TABLE_ECAM_FILE = 4

eNC_TABLE_ECAM_ARRAY = 5

eNC_TABLE_MAX = 6

This enumeration is used as the input for these functions, to distinguish between ECAM and PVT. Currently only eNC_TABLE_PVT_FILE and eNC_TABLE_PVT_ARRAY can be applied.

ulMaxNumberOfPoints

The maximum number of points allocated. Theoretical maximum value of $4.294E^{+9}$, however in practice will be limited well below this value.

ulUnderflowThreshold

The underflow limit. Cannot be larger than the value for ulMaxNumberOfPoints.

usAxisRef

Axis/group reference handle type returned by GetAxisRef command.

usDimension

The dimensions of the table. A +ve number

ucIsDynamicMode

Boolean result to question, Is the mode Dynamic or not? 0, or 1

ucIsPosAbsolute

Boolean result to question, Is the position Absolute or not? 0, or 1.

This parameters is for future use.

ucIsCyclic

Boolean result to question, Is the mode Cyclic or not? 0, or 1

MMC_INITTABLE_OUT Structure

```
typedef struct mmc_inittable_out{
MC_PATH_REF hMemHandle;
unsigned short usStatus;
short usErrorID;
} MMC_INITTABLE_OUT;
```

Parameters

hMemHandle

MC_PATH_REF alias for unsigned integer with values and handle to a journal entry where the pointer to the shared memory is located, and indicates the memory area where the spline is allocated. In fact this is the unique identifier between PathSelect, MovePath and PathUnselect commands. MC_PATH_REF is the journal entry path reference.

hMemHandle produces +ve Integer values.

usStatus

Bitwise returned command status with the values MMC_REMOTE_FUNC_STATUS_BIT_ERROR or 0 (OK).

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 9-1 describes the function for MMC_InitTable.

Figure 9-1: MMC_InitTable function

9.7.1.2. Function Code Example

Refer to the example in section 9.7.6.2

9.7.2. MMC_LoadTableFromFile

The function allocates a memory segment in the G-MAS shared memory according to the dimension and number of points given in a file.

```
MMC_LIB_API int MMC_LoadTableFromFileCmd(
MMC_CONNECT_HNDL hConn,
MMC_LOADTABLEFROMFILE_IN* pInParam,
MMC_LOADTABLE_OUT* pOutParam
);
```

Motion Mode	NC – Supported	Distributed – Not supported
Source	GMAS\includes\MMC_general_API.h GMAS\includes\MMC_PVT_ECAM_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_LOADTABLEFROMFILE_IN** input data structure using the MMC_LoadTableFromFileCmd.

pOutParam

Points to the **MMC_LOADTABLE_OUT** output structure receiving information, as a result of calling the MMC_LoadTableFromFileCmd.

Remarks

The file always contains data.

Scope

All

MMC_LOADTABLEFROMFILE_IN Structure

```
typedef struct mmc_loadtablefromfile_in{
NC_MOTION_TABLE_TYPE_ENUM eTableType;
unsigned short usAxisRef;
MC_PATH_DATA_REF pPathToTableFile;
} MMC_LOADTABLEFROMFILE_IN;
```

Parameters

eTableType

The Table type defined according to the enumerator parameter NC_MOTION_TABLE_TYPE_ENUM defined by:

eNC_TABLE_NONE = 0

eNC_TABLE_SPLINE = 1

eNC_TABLE_PVT_FILE = 2

eNC_TABLE_PVT_ARRAY = 3

eNC_TABLE_ECAM_FILE = 4

eNC_TABLE_ECAM_ARRAY = 5

eNC_TABLE_MAX = 6

This enumeration is used as the input for these functions, to distinguish between ECAM and PVT. Currently only eNC_TABLE_PVT_FILE and eNC_TABLE_PVT_ARRAY can be applied.

usAxisRef

Axis/group reference handle type returned by GetAxisRef command. Any +ve value.

pPathToTableFile

Unix path to the file that should be loaded

MC_PATH_DATA_REF Where the enumerator MC_PATH_DATA_REF describes the I/O definition of the path data reference using the array [NC_PVT_ECAM_MAX_ARRAY_SIZE] that defines the maximum array size file path data.

MC_PATH_DATA_REF can have values of any characters.

NC_PVT_ECAM_MAX_ARRAY_SIZE is 170.

MMC_LOADTABLE_OUT Structure

```
typedef struct MMC_LOADTABLE_OUT {
unsigned short usStatus;
short usErrorID;
MC_PATH_REF hMemHandle;
} MMC_LOADTABLE_OUT;
```

Parameters

hMemHandle

MC_PATH_REF alias for unsigned integer with values and handle to a journal entry where the pointer to the shared memory is located, and indicates the memory area where the spline is allocated. In fact this is the unique identifier between PathSelect, MovePath and PathUnselect commands. MC_PATH_REF is the journal entry path reference.

hMemHandle produces +ve Integer values.

usStatus

Bitwise returned command status with the values MMC_REMOTE_FUNC_STATUS_BIT_ERROR or 0 (OK).

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Displays an error code as -ve or +ve integers.

Figure 9-2 describes the function for MMC_LoadTableFromFile.

Figure 9-2: MMC_LoadTableFromFile function

9.7.2.2. Function Code Example

Refer to the example in section 9.7.6.2

9.7.3. MMC_UnloadTable

The function unloads a table from the G-MAS and frees a memory segment in the G-MAS shared memory according to the dimension and number of points given in a file.

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

Motion Mode	NC – Supported	Distributed – Not supported
Source	GMAS\includes\MMC_general_API.h GMAS\includes\MMC_PVT_ECAM_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_UNLOADTABLE_IN** input data structure using the MMC_UnloadTableFromFileCmd.

pOutParam

Points to the **MMC_UNLOADTABLE_OUT** output structure receiving information, as a result of calling the MMC_UnloadTableFromFileCmd.

Remarks

The file always contains data.

Scope

All

MMC_UNLOADTABLE_IN Structure

typedef struct mmc_unloadtable_in{
MC_PATH_REF hMemHandle;
unsigned char ucExecute;
}MMC_UNLOADTABLE_IN

Parameters

hMemHandle

MC_PATH_REF enumerator handle to a journal entry where the pointer to the shared memory is located, obtained from the PathSelect command. MC_PATH_REF is the journal entry path reference.

hMemHandle can have Integer values.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values. Currently not in use and set to 1.

MMC_UNLOADTABLE_OUT Structure

typedef struct mmc_unloadtable_out{
unsigned short usStatus;
short usErrorID;
}MMC_UNLOADTABLE_OUT

Parameters

usStatus

Bitwise returned command status with the values MMC_REMOTE_FUNC_STATUS_BIT_ERROR or 0 (OK).

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 9-2 describes the function for MMC_UnloadTable.

Figure 9-3: MMC_UnloadTable function

9.7.3.2. Function Code Example

Refer to the example in section 9.7.6.2

9.7.4. MMC_MoveTable

This function moves the Table along a selected path.

```
MMC_LIB_API int MMC_MoveTableCmd(
MMC_CONNECT_HNDL hConn,
MMC_AXIS_REF_HNDL hAxisRef,
MMC_MOVETABLE_IN* pInParam,
MMC_MOVETABLE_OUT* pOutParam
);
```

Motion Mode	NC – Supported	Distributed – Not supported
Source	GMAS\includes\MMC_general_API.h GMAS\includes\MMC_PVT_ECAM_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command.

pInParam

Points to the **MMC_MOVETABLE_IN** input data structure using the MMC_MoveTableCmd.

pOutParam

Points to the **MMC_MOVETABLE_OUT** output structure receiving information, as a result of calling the MMC_MoveTableCmd.

Remarks

When applying to a single-axis PVT, the multi-axes values of the parameter MMC_MOVETABLE_OUT are not relevant, and therefore any value can be provided.

Scope

All

MMC_MOVETABLE_IN Structure

```
typedef struct MMC_MOVETABLE_IN{
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
MC_PATH_REF hMemHandle;
unsigned char ucSuperImposed;
unsigned char ucExecute;
} MMC_MOVETABLE_IN;
```

Parameters

fTransitionParameter [NC_MAX_NUM_AXES_IN_NODE]

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section **5.8.2.1 Transition Modes.**

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eCoordSystem

Define the types of supported coordinate systems using the MC_COORD_SYSTEM_ENUM enumerator value. The options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

eTransitionMode

Define the supported NC_TRANSITION_MODE_ENUM enumerator transition modes. Refer to the section **5.8.2.1 Transition Modes**.

The options are:

MC_TM_NONE_MODE = 0, only this mode supported at this time

MC_TM_MAX_VELOCITY_MODE = 1, Not supported at this time

MC_TM_DEFINED_VELOCITY_MODE = 2, Not supported at this time

MC_TM_CORNER_DISTANCE_MODE = 3, Not supported at this time

MC_TM_MAX_CORNER_DEVIATION_MODE = 4, Not supported at this time

MC_TM_SWITCH_RADIUS_MODE = 5, Not supported at this time

eBufferMode

The MC_BUFFERED_MODE_ENUM enumerator defines the behavior of the axis. Modes are as follows, but only the Buffered Mode is supported:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2, only this mode supported
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately. The buffer is cleared

Buffered The next function block affects the axis as soon as the previous movement is completed.

BlendingLow The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).

BlendingPrevious Blending with the velocity of function block 1 at the end-position of this block

BlendingNext Blending with the velocity of function block 2 at end-position of function block1

BlendingHigh Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

hMemHandle

MC_PATH_REF enumerator handle to a journal entry where the pointer to the shared memory is located obtained from the PathSelect command. MC_PATH_REF is the journal entry path reference.

hMemHandle has Integer values.

ucSuperimposed

Whether the option to superimpose is operated or not. Values accepted are Boolean TRUE/FALSE. Not currently in use.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVETABLE_OUT Structure

```
typedef struct MMC_MOVETABLE_OUT {
unsigned int uiHandle;
unsigned short usStatus;
short usErrorID;
} MMC_MOVETABLE_OUT;
```

Parameters

uiHandle

Handle to a journal entry where the pointer to the shared memory is located, and indicates the memory area where the spline is allocated. In fact this is the unique identifier between PathSelect, MovePath and PathUnselect commands. MC_PATH_REF is the journal entry path reference.

uiHandle produces +ve Integer values.

usStatus

Bitwise returned command status with the values MMC_REMOTE_FUNC_STATUS_BIT_ERROR or 0 (OK).

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 9-4 describes the function for MMC_MoveTable

Figure 9-4: MMC_MoveTable function

9.7.4.2. Function Code Example

Refer to the example in section 9.7.6.2

9.7.5. MMC_AppendPointsToTable

This function appends pointa to an existing table.

```
int MMC_AppendPointsToTableCmd(
MMC_CONNECT_HNDL hConn,
MMC_APPENDPOINTSTOTABLE_IN* pInParam,
MMC_APPENDPOINTSTOTABLE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Not Supported
Source	GMAS\includes\MMC_general_API.h GMAS\includes\MMC_PVT_ECAM_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_APPENDPOINTSTOTABLE_IN** input data structure using the MMC_AppendPointsToTableCmd function.

pOutParam

Points to the **MMC_UNLOADTABLE_OUT** output structure receiving information, as a result of calling the MMC_AppendPointsToTableCmd function.

Remarks

None

Scope

All

MMC_APPENDPOINTSTOTABLE_IN Structure

```
typedef struct mmc_appendpointstotable_in{
double dTable[NC_PVT_ECAM_MAX_ARRAY_SIZE];
NC_MOTION_TABLE_TYPE_ENUM eTableType;
MC_PATH_REF hMemHandle;
unsigned long ulStartIndex;
unsigned long ulNumberOfPoints;
unsigned short usAxisRef;
unsigned char ucIsTimeAbsolute;
unsigned char ucIsAutoAppend;
} MMC_APPENDPOINTSTOTABLE_IN;
```

Parameters

dTable

Range of Points to be added to the Table, with 8 byte ± values accepted.

The array consists of a range of points in columns, where each three cells represents one point described by its Position, Velocity, and Time.

The maximum value of the NC_PVT_ECAM_MAX_ARRAY_SIZE array variable is 170. This is the maximum number of point allowed in the table.

eTableType

The Table type defined according to the enumerator parameter NC_MOTION_TABLE_TYPE_ENUM defined by:

eNC_TABLE_NONE = 0

eNC_TABLE_SPLINE = 1

eNC_TABLE_PVT_FILE = 2

eNC_TABLE_PVT_ARRAY = 3

eNC_TABLE_ECAM_FILE = 4

eNC_TABLE_ECAM_ARRAY = 5

eNC_TABLE_MAX = 6

This enumeration is used as the input for these functions, to distinguish between ECAM and PVT. Currently only eNC_TABLE_PVT_FILE and eNC_TABLE_PVT_ARRAY can be applied.

hMemHandle

MC_PATH_REF enumerator handle to a journal entry where the pointer to the shared memory is located, obtained from the PathSelect command. MC_PATH_REF is the journal entry path reference.

hMemHandle can have Integer values.

ulStartIndex

Integer value of the index start. Any +ve value.

ulNumberOfPoints

Number of points to be appended to table. Any +ve number. The value cannot exceed MaxNumberOfPoints used in the function MMC_InitTable.

usAxisRef

Axis/group reference handle type returned by GetAxisRef command. Any +ve value.

ucIsTimeAbsolute

Boolean result to the question, Is the Time Absolute or not? Select the mode of the parameter:

0 - absolute mode
Every "time" input is used as absolute time when the current point will end the motion and reach the desired position.

1 - relative mode
Every "time" input is used as the time that will take to move from previous point to the end of current point.

ucIsAutoAppend

Boolean result to the question, Is the Append operation to be automatic or not? 0, or 1.

MMC_APPENDPOINTSTOTABLE_OUT Structure

```
typedef struct mmc_appendpointstotable_out{
unsigned short usStatus;
short usErrorID;
} MMC_APPENDPOINTSTOTABLE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the values MMC_REMOTE_FUNC_STATUS_BIT_ERROR or 0 (OK).

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 9-5 describes the function for MMC_AppendPointsToTable.

Figure 9-5: MMC_AppendPointsToTable function

9.7.5.2. Function Code Example

Refer to the example in section 9.7.6.2

9.7.6. MMC_GetTableIndex

This function obtains the PVT index.

```
int MMC_GetTableIndexCmd(
MMC_CONNECT_HNDL hConn,
MMC_GETTABLEINDEX_IN* pInParam,
MMC_GETTABLEINDEX_OUT* pOutParam
);
```

Motion Mode	NC – Supported	Distributed – Not supported
Source	GMAS\includes\MMC_general_API.h GMAS\includes\MMC_PVT_ECAM_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command.

pInParam

Points to the **MMC_GETTABLEINDEX_IN** input data structure using the MMC_GetTableIndexCmd.

pOutParam

Points to the **MMC_GETTABLEINDEX_OUT** output structure receiving information, as a result of calling the MMC_GetTableIndexCmd.

Remarks

When applying to a single-axis PVT, the multi-axes values of the parameter MMC_MOVETABLE_OUT are not relevant, and therefore any value can be provided.

Scope

All

MMC_GETTABLEINDEX_IN Structure

```
typedef struct mmc_gettableindex_in{
init MC_PATH_REF hMemHandle;
} MMC_GETTABLEINDEX_IN;
```

Parameters

MC_PATH_REF hMemHandle

MC_PATH_REF enumerator handle to a journal entry where the pointer to the shared memory is located. MC_PATH_REF is the journal entry path reference.

hMemHandle can have integer values.

MMC_GETTABLEINDEX_OUT Structure

```
typedef struct mmc_gettableindex_out{
unsigned long ulCurrentIndex;
unsigned short usStatus;
short usErrorID;
} MMC_GETTABLEINDEX_OUT;
```

Parameters

ulCurrentIndex

Obtain the current index. Any +ve number.

usStatus

Bitwise returned command status with the values MMC_REMOTE_FUNC_STATUS_BIT_ERROR or 0 (OK).

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 9-6 describes the function block for MMC_GetTableIndex.

Figure 9-6: MMC_GetTableIndex function

9.7.6.2. Function Code Example

```
MMC_INITTABLE_IN     stInitTableIn;
MMC_INITTABLE_OUT    stInitTableOut;

MMC_LOADTABLEFROMFILE_IN stLoadTableFromFileIn;
MMC_LOADTABLE_OUT stLoadTableOut;

MMC_APPENDPOINTSTOTABLE_IN stAppendPointsIn;
MMC_APPENDPOINTSTOTABLE_OUT stAppendPointsOut;

MMC_GETTABLEINDEX_IN stGetTableIndexIn;
MMC_GETTABLEINDEX_OUT stGetTableIndexOut;

MMC_MOVETABLE_IN stMoveTableIn;
MMC_MOVETABLE_OUT stMoveTableOut;

stInitTableIn.eTableType = eNC_TABLE_PVT_ARRAY; //The table is defined as a PVT array
stInitTableIn.hAxisRef = aRef; //Group axes reference returned by the GatAxisRef command
stInitTableIn.ucIsCyclic = 1; //Mode is cyclic
stInitTableIn.ucIsPosAbsolute = 1; //Position is Absolute
stInitTableIn.ucIsTimeAbsolute = 0; //Time is not Absolute - This parameter is not
available for the function
stInitTableIn.usDimension = 2; //Dimensions of the table
stInitTableIn.usMaxNumberOfPoints = 50; //Maximum points allocated to the table

rc = MMC_InitTableCmd(hConn, &stInitTableIn, &stInitTableOut);
if (NC_OK != rc)
{
      HandleError();
}


stAppendPointsIn.eTableType = eNC_TABLE_PVT_ARRAY; //The table is defined as a PVT array
stAppendPointsIn.hMemHandle = stInitTableOut.hMemHandle; //Pointer to a journal entry
where the pointer is stored
stAppendPointsIn.usStartIndex = 0; //Value of the start index

FillTable(stLoadTableFromArrayIn.dTable);

rc = MMC_AppendPointsToTableCmd(hConn, &stAppendPointsIn, &stAppendPointsOut);
if (NC_OK != rc)
{
      HandleError();
```

```
	}

	stLoadTableFromFileIn.eTableType = eNC_TABLE_PVT_FILE; //The table is defined as a PVT
file
	stLoadTableFromFileIn.hAxisRef = aRef; //Axis reference returned by the GatAxisRef
command
	strcpy(stLoadTableFromFileIn.pPathToTableFile, pFileName); //Unix path to the file to be
loaded

	rc = MMC_LoadTableFromFileCmd(hConn, &stLoadTableFromFileIn, &stLoadTableOut);
	if (NC_OK != rc)
	{
		HandleError();
	}

	stMoveTableIn.eBufferMode = MC_BUFFERED_MODE; //standard buffered mode supported
	stMoveTableIn.eCoordSystem = MC_MCS_COORD; // MC_COORD_SYSTEM_ENUM enumerator value of
MCS coord system
	stMoveTableIn.eTransitionMode = MC_TM_NONE_MODE; // only transition mode supported
	stMoveTableIn.fTransitionParameter[0] = 0; // 0 Array value for specific transition
	stMoveTableIn.hMemHandle = stLoadTableOut.hMemHandle; // MC_PATH_REF enum handle
	stMoveTableIn.ucExecute = 1;  //
	stMoveTableIn.ucSuperImposed = 0;  //

	rc = MMC_MoveTableCmd(hConn, aRef, &stMoveTableIn, &stMoveTableOut);
	if (NC_OK != rc)
	{
		HandleError();
	}

	stGetTableIndexIn.hMemHandle = stInitTableOut.hMemHandle;

	rc = MMC_GetTableIndexCmd(hConn, &stGetTableIndexIn, &stGetTableIndexOut);
	if (NC_OK != rc)
	{
		HandleError();
	}
```

Chapter 10: API Services and Operations

This chapter describes the API services and operations for the G-MAS, and involves the following:

- Main configuration variables
- G-MAS Preoperational Mode. Refer to **Chapter** 2: **G-MAS Overview**, section on page **2-29** for further details
- EtherCAT Configuration Mode. Refer to **Chapter 2: G-MAS Overview**, section on page **2-29** for further details
- Data Recording. Refer to **Chapter 14:, Data Recording on page 14-1038** for further details
- Resource file uploading and downloading
- Download new firmware version

The following main configuration function blocks are described, where MMC_Connection_Param_Struct is an administrative function only:

Function Block	Services and Operation
MMC_InitConnection	Main configuration variables
MMC_IPCInitConnection	
MMC_RpcInitConnection	
MMC_CloseConnection	
MMC_CmdStatus	
MMC_Config	
MMC_Exit	
MMC_FreeFbStat	
MMC_GetAxisByName	
MMC_GetGroupByName	
MMC_GetVersion	
MMC_ResetMultiAxisControl	
MMC_SaveParam	
MMC_ShowNodeStat	
MMC_ClearNodeFbList	
MMC_CloseConnection	
MMC_CreateSYNCTimer	
MMC_DestroySYNCTimer	
MMC_DownloadFoE	
MMC_Dwell	

MMC_GetActiveVectorsNum	
MMC_GetErrorCodeDescriptionByID	
MMC_GetFoEStatus	
MMC_GetEthercatCommStatistics	
MMC_GetEnquireFbStatus	
MMC_GetGroupMembersInfo	
MMC_GetStatusRegister	
MMC_GetVersionEx	
MMC_LoadParam	
MMC_RpcInitConnectionEx	
MMC_SetEnquireFbStatus	
MMC_SetDefaultParameters	
MMC_SetDefaultParametersGlobal	
MMC_SetIsToLoadGlobalParams	
MMC_GetActiveAxesNum	
MMC_ToggleConsoleOutput	
MMC_GetCyclesCounter	
MMC_WriteGroupOfParameters	
MMC_ReadGroupOfParameters	
MMC_ChangeToPreOPMode	G-MAS Preoperational Mode
MMC_ChangeToOperationMode	
GetGMASOperationMode	
MMC_GetResList	Resource file variables involving list, snapshot, export, and import.
MMC_GetResSnapshot	
MMC_ResExportFile	
MMC_ResImportFile	
MMC_GetVerPath	
MMC_DownloadVersion	
MMC_ReadDownloadVersionStatus	
MMC_SetVerPath	

10.1. Download Firmware and Errors

The following download firmware function blocks are not detailed in this document, but will be necessary to download and update the firmware. For further details, refer to Elmo.

Function Block	Services and Operation
MMC_GetVerPath	Download firmware variables involving update and read firmware version.
MMC_DownloadFirmware	
MMC_ReadDownloadVersionStatus	
MMC_SetVerPath	

The following list describes the firmware download errors possible during an update operation.

Error Parameter	ID	Explanation
DOWNLOAD_VERSION_ERROR_FS	-1	Download Firmware. Target file system error
DOWNLOAD_VERSION_ERROR_TFTP	-2	Download Firmware. TFTP fail
DOWNLOAD_VERSION_ERROR_BURN	-3	Download Firmware. Copy to flash is fail.
DOWNLOAD_VERSION_ERROR_CRC	-4	Download Firmware. Wrong CRC.
DOWNLOAD_VERSION_NON_LINUX_IMAGE	-5	Download Firmware. Image is not Linux type.
DOWNLOAD_VERSION_ERROR_DF_IN_PROCESS	-6	Download Firmware. Error DF in process.
DOWNLOAD_VERSION_DIFFERENT_MD5SUM	-7	Wrong MD5 CHKSum
FALLBACK_VERSION_ERROR_CRC	-8	Fallback version has CRC error !!!
BOOTED_WITH_FALLBACK_VER_ERROR	-9	Booted with fallback error

10.2. Type Definitions

10.2.1. MMC_DOWNLOAD_TYPE_ENUM

The following resource enumerators define the download configuration resource file types for the variable MMC_DOWNLOAD_TYPE_ENUM.

ID	Parameters	Explanation
0	MMC_PARAMETER_FILE_DOWNLOAD	Download parameter file
1	MMC_RESOURCE_FILE_DOWNLOAD	Download resource file
2	MMC_SNAPSHOT_RESOURCE_FILE_DOWNLOAD	Download resource snapshot file
3	MMC_ETHERCAT_CFG_FILE_DOWNLOAD	Download Resource EtherCAT file
4	MMC_PERSONALITY_FILE_DOWNLOAD	Download G-MAS Personality file
5	MMC_USER_FILE_DOWNLOAD	Download G-MAS user file

10.2.2. MMC_CONNECT_HNDL

The following Connection Handle parameter types are defined with their maximum values.

Parameter	Max. Value	Explanation
NC_MAX_NUM_AXES_IN_NODE	16	Maximal number Axis in Group.
NODE_NAME_MAX_LENGTH	80	Maximal length of Axis/Group name.
NC_MAX_REC_SIGNALS_NUM	22	Maximal number recorder signals.
NC_MAX_REC_PARAMS_NUM	8	Maximal number recorder parameters.
NC_MAX_REC_PACKET_SIZE	1400	Maximal length of packet recorded data.
NC_MAX_LONG NC_MAX_REC_PACKET_SIZE	4	Maximal length of packet recorded data.
NC_NODES_SING_AXIS_NUM	96	
NC_NODES_MULT_AXIS_NUM	48	

10.3. Main Configuration Function Blocks

The following main configuration function blocks are described:

Main Configuration
MMC_ChangeToPreOPMode
MMC_ChangeToOperationMode
MMC_ClearNodeFbList
MMC_CmdStatus
MMC_CloseConnection
MMC_Config
MMC_CreateSYNCTimer
MMC_DestroySYNCTimer
MMC_DownloadFoE
MMC_Dwell
MMC_Exit
MMC_FreeFbStat
MMC_GetActiveVectorsNum
MMC_GetErrorCodeDescriptionByID
MMC_GetFoEStatus
MMC_GetEthercatCommStatistics
MMC_GetEnquireFbStatus
MMC_GetAxisByName
MMC_GetGroupByName
MMC_GetGroupMembersInfo
MMC_GetGMASOperationMode
MMC_GetStatusRegister
MMC_GetResList
MMC_GetResSnapshot
MMC_GetVersion

MMC_GetVersionEx
MMC_InitConnection
MMC_IPCInitConnection
MMC_LoadParam
MMC_RpcInitConnection
MMC_RpcInitConnectionEx
MMC_ResetMultiAxisControl
MMC_ResExportFile
MMC_ResImportFile
MMC_SaveParam
MMC_SetEnquireFbStatus
MMC_SetDefaultParameters
MMC_SetDefaultParametersGlobal
MMC_SetIsToLoadGlobalParams
MMC_ShowNodeStat
MMC_GetActiveAxesNum
MMC_ToggleConsoleOutput
MMC_GetCyclesCounter
MMC_WriteGroupOfParameters
MMC_ReadGroupOfParameters
MMC_GetVerPath
MMC_DownloadVersion
MMC_ReadDownloadVersionStatus
MMC_SetVerPath

10.3.1. MMC_ChangeToPreOPMode

Changes the G-MAS to preoperational mode.

```
MMC_LIB_API int MMC_ChangeToPreOPMode(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SET_GMAS_PREOP_IN* pInParam
OUT MMC_SET_GMAS_PREOP_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SET_GMAS_PREOP_IN** input data structure using the MMC_ChangeToPreOPMode function.

pOutParam

Points to the **MMC_SET_GMAS_PREOP_OUT** output structure receiving information, as a result of calling the MMC_ChangeToPreOPMode function.

Remarks

None

Scope

All

MMC_SET_GMAS_PREOP_IN Structure

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

Parameters

dummy

Dummy input. Any +ve character value.

MMC_SET_GMAS_PREOP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SET_GMAS_PREOP_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-1 describes the function block for MMC_ChangeToPreOPMode.

Figure 10-1: MMC_ChangeToPreOPMode function block

10.3.1.2. Function Block Code Example

```
int rc;
MMC_SET_GMAS_PREOP_IN       stSetGMASPreOp_in;
MMC_SET_GMAS_PREOP_OUT      stSetGMASPreOp_out;
//
// Inserting the structure parameters:
stSetGMASPreOp_in.ucDummy       = 1;     // Dummy input
//
rc = MMC_ChangeToPreOPMode (hConn, &stSetGMASPreOp_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.2. MMC_ChangeToOperationMode

Changes the G-MAS to operational mode.

```
MMC_LIB_API int MMC_ChangeToOperationMode(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SET_GMAS_OP_IN* pInParam
OUT MMC_SET_GMAS_OP_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SET_GMAS_OP_IN** input data structure using the MMC_ChangeToOperationMode function.

pOutParam

Points to the **MMC_SET_GMAS_OP_OUT** output structure receiving information, as a result of calling the MMC_ChangeToOperationMode function.

Remarks

None

Scope

All

MMC_SET_GMAS_OP_IN Structure

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

Parameters

dummy

Dummy input. Any +ve character value.

MMC_SET_GMAS_OP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SET_GMAS_OP_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-2 describes the function block for MMC_ChangeToOperationMode.

Figure 10-2: MMC_ChangeToOperationMode function block

10.3.2.2. Function Block Code Example

```
int rc;
MMC_SET_GMAS_OP_IN          stSetGMASOp_in;
MMC_SET_GMAS_OP_OUT         stSetGMASOp_out;
//
// Inserting the structure parameters:
stSetGMASOp_in.ucDummy      = 1;      // Dummy input
//
rc = MMC_ChangeToOperationMode (hConn, &stSetGMASOp_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.3. MMC_ClearNodeFbList

This adds the ability to clear the function block list of a specific node, i.e. either Axis or Group.This can only be performed if the node is not in a moving state.

```
int MMC_ClearNodeFbListCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_CLEARFBLIST_IN* pInParam,
OUT MMC_CLEARFBLIST_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_CLEARFBLIST_IN** input data structure using the MMC_ClearNodeFbList function.

pOutParam

Points to the **MMC_CLEARFBLIST_OUT** output structure receiving information, as a result of calling the MMC_ClearNodeFbList function.

Remarks

Refer to the use of the function in section **10.3.13 MMC_GetActiveVectorsNum on page 10-569**.

Scope

All

MMC_CLEARFBLIST_IN Structure

```
typedef struct mmc_clearfblist_in{
unsigned short usAxisRef;
}MMC_CLEARFBLIST_IN;
```

Parameters

usAxisRef

Axis reference. Any +ve bitwise integer.

MMC_CLEARFBLIST_OUT Structure

```
typedef struct mmc_clearfblist_out{
unsigned short usStatus;
short usErrorID;
}MMC_CLEARFBLIST_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-3 describes the function block for MMC_ClearNodeFbList

Figure 10-3: MMC_ClearNodeFbList function block

10.3.3.2. Function Block Code Example

10.3.4. MMC_CmdStatus

Sends a Read Function Block Status command to the G-MAS server for specific Axis/Group and receive status back.

```
MMC_LIB_API int MMC_CmdStatus(
IN MMC_CONNECT_HNDL hConn,
IN MMC_FBSTATUS_IN* pInParam,
OUT MMC_FBSTATUS_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_FBSTATUS_IN** input data structure using the MMC_CmdStatus function.

pOutParam

Points to the **MMC_FBSTATUS_OUT** output structure receiving information, as a result of calling the MMC_CmdStatus function.

Remarks

None

Scope

All

MMC_FBSTATUS_IN Structure

```
typedef struct{
unsigned int uiHndl;
}MMC_FBSTATUS_IN;
```

Parameters

uiHndl

Function block handle. Any +ve integer value

MMC_FBSTATUS_OUT Structure

```
typedef struct{
unsigned int uiFbStatus;
unsigned short usStatus;
short usErrorID;
unsigned short usFbErrorID;
}MMC_FBSTATUS_OUT;
```

Parameters

uiFbStatus

Returned function block status. Any +ve integer bitwise value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usFbErrorID

Returned function block error ID. Signals where a function block error occurs. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-3 describes the function block for MMC_CmdStatus

Figure 10-4: MMC_CmdStatus function block

10.3.4.2. Function Block Code Example

```
int rc;
MMC_FBSTATUS_IN        stFBStatus_in;
MMC_FBSTATUS_OUT       stFBStatus_out;
//
// Inserting the structure parameters:
stFBStatus_in.uiHndl   = 1;          //Function block handle
//
rc = MMC_CmdStatus (hConn, &stFBStatus_in, &stFBStatus_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.5. MMC_CloseConnection

Closes the connection to the G-MAS.

```
MMC_LIB_API int MMC_CloseConnection(
IN MMC_CONNECT_HNDL hConn
);
```

Motion Mode NC – Not relevant Distributed – not relevant

Source GMAS\includes\MMC_general_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

Remarks

None

Scope

The input parameter *hConn* is dependent on the connection handle value created when performing the InitConnection function. This value should therefore be retained with other connection handle thread values for use when a connection is to be closed.

Figure 10-5 describes the function block for MMC_CloseConnection

Figure 10-5: MMC_CloseConnection function block

10.3.5.1. Function Block Code Example

```
int rc;
//
// Inserting the structure parameters:
hConn         = 1 ;         // Connection Handle Type. Number from the connection handle
//
rc = MMC_CloseConnection (hConn);
printf("Connection State[%ld]\n", (long int)(MMC_CONNECT_HNDL) hConn);
if (rc != 0)
printf("ERROR:%s: MMC_CloseConnection fail\n", __func__);
{
    HandleError();
}
```

10.3.6. MMC_Config

Set G-MAS to configuration mode and allow changes to any configuration parameters.

```
MMC_LIB_API int MMC_ConfigCmd
(IN MMC_CONNECT_HNDL hConn,
IN MMC_CONFIG_IN* pInParam,
OUT MMC_CONFIG_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_CONFIG_IN** input data structure using the MMC_Config function.

pOutParam

Points to the **MMC_CONFIG_OUT** output structure receiving information, as a result of calling the MMC_Config function.

Remarks

There are two G-MAS operational modes:

- Normal
- Configuration

When any communication configuration parameters (network IP etc.) are changed using the Set command, this function is invoked to exit the configuration mode to the normal operational mode of the G-MAS.

Scope

All

MMC_CONFIG_IN Structure

```
typedef struct mmc_config_in{
unsigned char dummy;
}MMC_CONFIG_IN;
```

Parameters

dummy

Dummy character. Any +ve value accepted.

MMC_CONFIG_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIG_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-6 describes the function block for MMC_Config.

Figure 10-6: MMC_Config function block

10.3.6.2. Function Block Code Example

```
int rc;
MMC_CONFIG_IN     stConfig_in;
MMC_CONFIG_OUT    stConfig_out;
//
// Inserting the structure parameters:
stConfig_in.dummy = 1;     //dummy value
//
rc = MMC_ConfigCmd (hConn, &stConfig_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.7. MMC_CreateSYNCTimer

Creates a SYNC timer to synchronize servo-drive, G-MAS movements using the connection handle operator.

```
MMC_LIB_API int MMC_CreateSYNCTimer(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SYNC_TIMER_CB_FUNC func,
IN unsigned short usSYNCTimerTime
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

func

[IN] Points to the callback function MMC_SYNC_TIMER_CB_FUNC using MMC_CreateSYNCTimer.

usSYNCTimerTime

[IN] Defines the time between which a synchronization message is sent as an event.

Remarks

None

Scope

All

Figure 10-7 describes the function for MMC_CreateSYNCTimer.

Figure 10-7: MMC_CreateSYNCTimer function

10.3.7.1. Function Block Code Example

10.3.8. MMC_DestroySYNCTimer

Removes the SYNC timer to synchronize servo-drive, G-MAS movements using the connection handle operator.

```
MMC_LIB_API int MMC_DestroySYNCTimer(
IN MMC_CONNECT_HNDL hConn
);
```

Motion Mode NC - Supported Distributed - Supported

Source GMAS\includes\MMC_general_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

Remarks

None

Scope

All

Figure 10-8 describes the function for MMC_DestroySYNCTimer.

Figure 10-8: MMC_DestroySYNCTimer function

10.3.8.1. Function Code Example

10.3.9. MMC_DownloadFoE

Manages downloads of a fileor files over EtherCAT to the G-MAS.

Important: **To use this function refer to Elmo for support.**

```
MMC_LIB_API int MMC_DownloadFoE(
IN MMC_CONNECT_HNDL hConn,
IN MMC_DOWNLOADFOE_IN* pInParam,
OUT MMC_DOWNLOADFOE_OUT* pOutParam);
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_DOWNLOADFOE_IN** input data structure using the MMC_DownloadFoE function.

pOutParam

Points to the **MMC_DOWNLOADFOE_OUT** output structure receiving information, as a result of calling the MMC_DownloadFoE function.

Remarks

The FoE download procedure in the GMAS, is performed for N slaves on the bus in parallel. The master moves the slaves to Bootstrap mode and sends periodically at each Mailbox cycle, a full Mailbox message to each slave containing relevant sections of the file, until the complete file is downloaded to all N slaves.

Scope

EtherCAT

MMC_DOWNLOADFOE_IN Structure

```
t typedef struct mmc_downloadfoe_in{
unsigned short pwSlaveId[NC_NODES_SING_AXIS_NUM];
char pcFileName[256];
unsigned char pucServer[4];
unsigned char ucSlavesNum;
}MMC_DOWNLOADFOE_IN;
```

Parameters

pwSlaveId[NC_NODES_SING_AXIS_NUM]

Slave ID to which the download is sent, with a limit of 3 characters with any +ve value, dependant on the array [NC_NODES_SING_AXIS_NUM], the number of single axis nodes.

pcFileName[256]

Full location and filename to be downloaded to the G-MAS from the host, with a limit of 256 characters

pucServer[4]

Serverhost IP with a limit of four characters (32 bit)

ucSlavesNum

Number of slaves to download the files with a maximum of 76 slaves.

MMC_DOWNLOADFOE_OUT Structure

```
typedef struct mmc_downloadfoe_out{
unsigned short usStatus;
unsigned short usErrorID;
}MMC_DOWNLOADFOE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block.

Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-9 describes the function for MMC_DownloadFoE.

Figure 10-9: MMC_DownloadFoE function

10.3.9.2. Function Code and Implementation Example

```
void DownloadFoe()
{
     MMC_DOWNLOADFOE_IN dlfoe ;
     MMC_DOWNLOADFOE_OUT dlfoeout ;
     MMC_GETFOESTATUS_OUT foestat ;
     MMC_GET_GMASOP_MODE_OUT pOpmode ;

     MMC_GETCOMMSTATISTICSEX_IN gcstat_In ;
     MMC_GETCOMMSTATISTICSEX_OUT gcstat_Out ;
     int i ;
     //
     //
     // Before DownloadingFOE - It is good practice that drives will be reset because
     // if one of the drives is after DownloadFoE and was not reset, its state and statitstics
are unknown.
     //
     dlfoe.pwSlaveId[0]=0 ;      // Note: Slave ID is inserted here !!
     dlfoe.pwSlaveId[1]=1 ;      // Note: Slave ID is inserted here !!
     //
     dlfoe.ucSlavesNum = 2;      // Number of relevant slaves in the pwSlaveId array.
     //
     // Same for slave statistics:
     gcstat_In.pwSlaveId[0] = 0 ;
     gcstat_In.pwSlaveId[1] = 1 ;
     gcstat_In.ucSlavesNum = 2 ;
     //
     // Insert IP of tftp server. Usually the connection IP of the PC.
     dlfoe.pucServer[0] = 10 ;
     dlfoe.pucServer[1] = 10 ;
     dlfoe.pucServer[2] = 20 ;
     dlfoe.pucServer[3] = 55 ;
     //
     // Copy file path name to the structure. Should be relative to the tftp folder
     strcpy(dlfoe.pcFileName,"FoEFW  01.01.04.68 27Oct2011P01G.abs") ;

     // Start tftp server on host. Only then call the MMC_DownloadFoE.
     //
     int rc = MMC_DownloadFoE(conn_hndl,&dlfoe,&dlfoeout)  ;
     if(rc < 0)
     {
          // Error Calling MMC_DownloadFoE. Error in dlfoeout.usErrorID
          return ;
     }
```

```
    //
    // If we reached this line, the tftp was succesful. Poll the GMAS for results:
    while(TRUE)
    {
        Sleep(100) ;
        //
        // Check the FoE progress:
        MMC_GetFoEStatus(conn_hndl,&foestat);
        if(rc < 0)
        {
            // Error Calling MMC_GetFoEStatus. Error in foestat.usErrorID
            return ;
        }
        //
        // Check that the FoE started.
        if(foestat.ucFOEStarted)
        {
            rc = MMC_GetGMASOperationMode(conn_hndl,&pOpmode) ;
            if(rc < 0)
            {
                // Error Calling MMC_GetGMASOperationMode. Error in pOpmode.usErrorID
                return ;
            }
            // Print Remaining time - foestat.ucProgress
            //
            // Check Foe Download progress is over and GMAS back in operational mode.
            if ((foestat.ucProgress == 0) && (pOpmode.ucResult == 0))
            {
                //
                // Download over. Check to see if any drives failed.
                for(i = 0 ; i < dlfoe.ucSlavesNum ; i++)
                {
                    if(foestat.pstSlavesErrorID[i].sErrorID != 0)
                    {
                        // Error on one of the slaves. Print error:
                        // SlaveID: foestat.pstSlavesErrorID[i].usSlaveID  has error -
foestat.pstSlavesErrorID[i]
                    }
                }
                // Notify user to switch drives Off / On and then check the download status.
wait 5 sec's.
                //
                // please note - MAX 76 slaves can be read.
                rc = MMC_GetEthercatCommStatistics(conn_hndl,&gcstat_In,&gcstat_Out) ;
                if(rc < 0)
                {
                    // Error Calling MMC_GetEthercatCommStatistics. Error in
gcstat_Out.usErrorID
                    return ;
                }
                //
                // gcstat_Out.ucMasterState - Should be EcatStateO  . operational.
                // Good idea to read number of slaves on bus - gcstat_Out.usNumOfSlaves
                // gcstat_Out.ucMasterDiagnosticState - All bits should be 0, except for:
EcatMasterDiagnosticStateUpdated, EcatMasterDiagnosticStateDefaultDataWasSet bits.
                //
                for(i = 0 ; i < gcstat_In.ucSlavesNum ; i++)
                {
                    // gcstat_Out.pucAxesState[i] - Should be EcatStateO.
                    // gcstat_Out.pucAxesDiagnosticState[i] - Should be 0.
                    if((gcstat_Out.pstSII_Content[i].ulVendorId == 0x9A) &&
(gcstat_Out.pstSII_Content[i].ulRevisionNo <= 0xFF))
                    {
                        //
                        // Drive stuck in no firmware state. Notify User. In This case the
InitCmdFail in diagnostics is not relevant.
                    }
                }
                return ;
```

```
                }
            }
        }
}

int OnConnectGetDiagnostics()
{
    int rc ;
    MMC_GET_GMASOP_MODE_OUT pOpmode ;

    MMC_GETCOMMSTATISTICSEX_IN gcstat_In ;
    MMC_GETCOMMSTATISTICSEX_OUT gcstat_Out ;
    int i ;
    //
    rc = MMC_GetGMASOperationMode(conn_hndl,&pOpmode) ;
    if(rc < 0)
    {
        // Error Calling MMC_GetGMASOperationMode. Error in pOpmode.usErrorID
        return ;
    }
    //
    // Check GMAS Operational state. If == 2, then in Download FOE state.
    if (pOpmode.ucResult == 2)
    {
        // GMAS in Download FoE state. We decided that a mesage will be shown to user that
the GMAS is in Download FoE.
    }

    rc = MMC_GetEthercatCommStatistics(conn_hndl,&gcstat_In,&gcstat_Out) ;
    if(rc < 0)
    {
        // Error Calling MMC_GetEthercatCommStatistics. Error in gcstat_Out.usErrorID
        return ;
    }
    //
    // gcstat_Out.ucMasterState - Should be EcatStateO. operational.
    // Good idea to read number of slaves on bus - gcstat_Out.usNumOfSlaves. Should be
identical to number of drives configured.
    // gcstat_Out.ucMasterDiagnosticState - All bits should be 0, except for:
EcatMasterDiagnosticStateUpdated, EcatMasterDiagnosticStateDefaultDataWasSet bits.
    //
    for(i = 0 ; i < gcstat_In.ucSlavesNum ; i++)
    {
        // gcstat_Out.pucAxesState[i] - Should be EcatStateO.
        //
        if((gcstat_Out.pstSII_Content[i].ulVendorId == 0x9A) &&
(gcstat_Out.pstSII_Content[i].ulRevisionNo <= 0xFF))
        {
            //
            // One of the Drives is stuck in no firmware state. Notify User to go to
diagnostics tab - NOT TO CONFIGURATOR.
            // In this case - the gcstat_Out.pucAxesDiagnosticState[i] InitCmd bit may be
set..
        }
        else
        {
            // pucAxesDiagnosticState[i] should be 0 !
        }

    }
}
```

10.3.10. MMC_Dwell

This function sends a temporary halt command to the G-MAS.

```
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
);
```

Motion Mode	NC – Supported	Distributed – Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_DWELL_IN** input data structure using the MMC_Dwell function.

pOutParam

Points to the **MMC_DWELL_OUT** output structure receiving information as a result of calling the MMC_Dwell function.

Remarks

None

Scope

All

MMC_DWELL_IN Structure

```
typedef struct MMC_DWELL_IN{
unsigned long ulDwellTimeMs;
} MMC_DWELL_IN;
```

Parameters

ulDwellTimeMs

The time during which the G-MAS rests in millisecs. Any +ve value.

MMC_DWELL_OUT Structure

```
typedef struct MMC_DWELL_OUT{
unsigned int uiHandle;
unsigned short usStatus;
short usErrorID;
} MMC_DWELL_OUT;
```

Parameters

uiHandle

Handle to a journal entry where the pointer to the shared memory is located, and indicates the memory area where the spline is allocated. In fact this is the unique identifier between PathSelect, MovePath and PathUnselect commands. MC_PATH_REF is the journal entry path reference.

uiHandle produces +ve Integer values.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-10 describes the function for MMC_Dwell

Figure 10-10: MMC_Dwell function

10.3.10.2. Function Block Code Example

10.3.11. MMC_Exit

Changes the G-MAS from configuration mode back to regular mode.

```
MMC_LIB_API int MMC_ExitCmd
(IN MMC_CONNECT_HNDL hConn,
IN MMC_EXIT_IN* pInParam,
OUT MMC_EXIT_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_EXIT_IN** input data structure using the MMC_Exit function.

pOutParam

Points to the **MMC_EXIT_OUT** output structure receiving information, as a result of calling the MMC_Exit function.

Remarks

There are two G-MAS operational modes:

- Normal
- Configuration

When any communication configuration parameters (network IP etc.) are changed using the Set command, this function is invoked to exit the configuration mode to the normal operational mode of the G-MAS.

Scope

All

MMC_EXIT_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_EXIT_IN;
```

Parameters

dummy

Dummy character. Any +ve value accepted.

MMC_EXIT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_EXIT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-11 describes the function block for MMC_Exit

Figure 10-11: MMC_Exit function block

10.3.11.2. Function Block Code Example

```
int rc;
MMC_EXIT_IN       stExit_in;
MMC_EXIT_OUT      stExit_out;
//
// Inserting the structure parameters:
stExit_in.dummy   = 1;     //Function block handle
//
rc = MMC_ExitCmd (hConn, &stExit_in, &stExit_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.12. MMC_FreeFbStat

Returns debug information that contains the number of free function blocks in the system.

```
MMC_LIB_API int MMC_FreeFbStatCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_FREEFBSTAT_IN* pInParam,
OUT MMC_FREEFBSTAT_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – Not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_FREEFBSTAT_IN** input data structure using the MMC_FreeFbStat function.

pOutParam

Points to the **MMC_FREEFBSTAT_OUT** output structure receiving information, as a result of calling the MMC_FreeFbStat function.

Remarks

None

Scope

All

MMC_FREEFBSTAT_IN Structure

```
typedef struct{
unsigned int uiHndl;
}MMC_FREEFBSTAT_IN;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

MMC_FREEFBSTAT_OUT Structure

```
typedef struct{
unsigned int uiFreeLargeFb;
unsigned int uiFreeMediumFb;
unsigned int uiFreeSmallFb;
unsigned short usStatus;
short usErrorID;
}MMC_FREEFBSTAT_OUT;
```

Parameters

uiFreeLargeFb

Number of free large size function blocks. Any +ve integer value.

uiFreeMediumFb

Number of free medium size function blocks. Any +ve integer value.

uiFreeSmallFb

Number of free small size function blocks. Any +ve integer value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-12 describes the function block for MMC_FreeFbStat

Figure 10-12: MMC_FreeFbStat function block

10.3.12.2. Function Block Code Example

```
int rc;
MMC_FREEFBSTAT_IN      stFreeFBStat_in;
MMC_FREEFBSTAT_OUT     stFreeFBStat_out;
//
// Inserting the structure parameters:
stFreeFBStat_in.uiHndl  = 10;   // Requested function block handle
//
rc = MMC_FreeFbStatCmd (hConn, &stFreeFBStat_in, &stFreeFBStat_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.13. MMC_GetActiveVectorsNum

Displays the number of active vectors (groups) attached and managed by the G-MAS.

```
MMC_LIB_API int MMC_GetActiveVectorsNum(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETACTIVEVECTORSNUM_IN* pInParam,
OUT MMC_GETACTIVEVECTORSNUM_OUT* pOutParam
);
```

Motion Mode	NC – Not Supported	Distributed – Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GETACTIVEAXESNUM_IN** input data structure using the MMC_GetActiveVectorsNum function.

pOutParam

Points to the **MMC_GETACTIVEAXESNUM_OUT** output structure receiving information as a result of calling the MMC_GetActiveVectorsNum function.

Remarks

This function will provide this basic information without opening the G-MAS Personality file.

Scope

All

MMC_GETACTIVEVECTORSNUM_IN Structure

```
typedef struct {
unsigned char dummy;
}MMC_GETACTIVEVECTORSNUM_IN;
```

Parameters

dummy

Any dummy values

MMC_GETACTIVEVECTORSNUM_OUT Structure

```
typedef struct {
int iActiveVectorsNum;
unsigned short usStatus;
short usErrorID;
}MMC_GETACTIVEVECTORSNUM_OUT;
```

Parameters

iActiveVectorsNum Provides the actives vectors in a group. +ve integer value.

usStatus

Bitwise returned command status with the following values:

Aborted

Done

CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 10-13 describes the function block for MMC_GetActiveVectorsNum

Figure 10-13: MMC_GetActiveVectorsNum function block

10.3.13.2. Function Block Code Example

10.3.14. MMC_GetErrorCodeDescriptionByID

This function receives an error\warning code and returns the description and resolution from the Personality file.

```
MMC_LIB_API int MMC_GetErrorCodeDescriptionByID(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETERRORCODEDESCRIPTIONBYID_IN* pInParam,
OUT MMC_GETERRORCODEDESCRIPTIONBYID_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETERRORCODEDESCRIPTIONBYID_IN** input data structure using the MMC_GetErrorCodeDescriptionByID function. The input [IN] receives the error\warning number.

pOutParam

Points to the **MMC_GETERRORCODEDESCRIPTIONBYID_OUT** output structure receiving information, as a result of calling the MMC_GetErrorCodeDescriptionByID function. The output [OUT] returns two strings, status and error id.

Remarks

None

Scope

Errors from the G-MAS Personality file.

MMC_GETERRORCODEDESCRIPTIONBYID_IN Structure

```
typedef struct mmc_getcodedescriptionbyid_in{
int iCode;
Char cType;
} MMC_GETERRORCODEDESCRIPTIONBYID_IN;
```

Parameters

iCode

Error and or warning code value. Any integer value which may be +ve or –ve

cType

The code type, which may be one of the following:

1 – GMAS code

2 – Drive emergency code

MMC_GETERRORCODEDESCRIPTIONBYID_OUT Structure

```
typedef struct mmc_getcodedescriptionbyid_out{
char pResolution[1100];
char pDescription[256];
unsigned short usStatus;
short usErrorID;
} MMC_GETERRORCODEDESCRIPTIONBYID_OUT;
```

Parameters

pResolution[1100]

Character value of the resolution with a maximum of 1100 characters

pDescription[256]

Character value of the description with a maximum value of 256 characters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-14 describes the function for MMC_GetErrorCodeDescriptionByID

Figure 10-14: MMC_GetErrorCodeDescriptionByID function

10.3.14.2. Function Code Example

10.3.15. MMC_GetFoEStatus

Obtains the File over EtherCAT status after a file download using MMC_DownloadFoE, from a host to the G-MAS.

Important: **To use this function refer to Elmo for support.**

```
MMC_LIB_API int MMC_GetFoEStatus(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETFOESTATUS_IN* pInParam,
OUT MMC_GETFOESTATUS_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETFOESTATUS_IN** input data structure using the MMC_GetFoEStatus function.

pOutParam

Points to the **MMC_GETFOESTATUS_OUT** output structure receiving information, as a result of calling the MMC_GetFoEStatus function.

Remarks

During a FoE download procedure, the user can periodically poll for statistics related to the FoE download procedure via MMC_GetFoEStatus(). The statistics obtainable are:

- Percentage progress of the FoE download procedure.
- Status parameter for each slave which participates in the FoE download procedure. If this status parameter is less than zero, an error occurrs for the relevant slave and this slave is excluded from the procedure.
- General status flag which indicates the current status of the FoE download procedure. As long as the download FoE procedure continues successfully, this parameter will be zero. However if the download procedure fails, then this parameter contains the specific error-ID causing the procedure to fail.
- FOE download's start flag (rising edge flag) – this flag indicates that the FoE download procedure (a totally asynchrony procedure), actually started.

Scope

EtherCAT

MMC_GETFOESTATUS_IN Structure

```
typedef struct mmc_getfoestatus_in{
unsigned char ucDummy;
}MMC_GETFOESTATUS_IN;
```

Parameters

ucDummy

Dummy data. +ve characters of any value.

MMC_GETFOESTATUS_OUT Structure

```
typedef struct mmc_getfoestatus_out{
unsigned short usStatus;
unsigned short usErrorID;
short sFOEStatus;
FOE_SLAVE_INFO pstSlavesErrorID[NC_NODES_SING_AXIS_NUM];
unsigned char ucNumOfSlaves;
unsigned char ucProgress ;
unsigned char ucFOEStarted ;
}MMC_GETFOESTATUS_OUT;
```

Parameters

sFOEStatus

FoE status value, either an error code or No Error

pstSlavesErrorID

The Slaves error ID structure, dependant on the array [NC_NODES_SING_AXIS_NUM], the number of single axis nodes.

FOE_SLAVE_INFO

```
typedef struct foe_slave_info{
unsigned short usSlaveID;
short sErrorID;
}FOE_SLAVE_INFO;
```

usSlaveID

Slave ID for the FoE

sErrorID

The error ID of the slave if a fault occurs.

ucNumOfSlaves

The number of slaves for which the file is downloaded, with a maximum of 76 slaves.

ucProgress

The progress of the FoE as a +ve percentage value.

ucFOEStarted

The FoE actual start position flag when the download starts (rising edge) as a +ve character value. Used when a number of file downloads occur from the server. Prevents the progress bar starting before the file download occurs.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-15 describes the function for MMC_GetFoEStatus

Figure 10-15: MMC_GetFoEStatus function

10.3.15.2. Function Code Example

```
void DownloadFoe()
{
    MMC_DOWNLOADFOE_IN dlfoe ;
    MMC_DOWNLOADFOE_OUT dlfoeout ;
    MMC_GETFOESTATUS_OUT foestat ;
    MMC_GET_GMASOP_MODE_OUT pOpmode ;

    MMC_GETCOMMSTATISTICSEX_IN gcstat_In ;
    MMC_GETCOMMSTATISTICSEX_OUT gcstat_Out ;
    int i ;
    //
    //
    // Before DownloadingFOE - It is good practice that drives will be reset because
    // if one of the drives is after DownloadFoE and was not reset, its state and statitstics
are unknown.
    //
    dlfoe.pwSlaveId[0]=0 ;     // Note: Slave ID is inserted here !!
    dlfoe.pwSlaveId[1]=1 ;     // Note: Slave ID is inserted here !!
    //
    dlfoe.ucSlavesNum = 2;     // Number of relevant slaves in the pwSlaveId array.
    //
    // Same for slave statistics:
    gcstat_In.pwSlaveId[0] = 0 ;
    gcstat_In.pwSlaveId[1] = 1 ;
    gcstat_In.ucSlavesNum = 2 ;
    //
    // Insert IP of tftp server. Usually the connection IP of the PC.
    dlfoe.pucServer[0] = 10 ;
    dlfoe.pucServer[1] = 10 ;
    dlfoe.pucServer[2] = 20 ;
    dlfoe.pucServer[3] = 55 ;
    //
    // Copy file path name to the structure. Should be relative to the tftp folder
    strcpy(dlfoe.pcFileName,"FoEFW  01.01.04.68 27Oct2011P01G.abs") ;

    // Start tftp server on host. Only then call the MMC_DownloadFoE.
    //
    int rc = MMC_DownloadFoE(conn_hndl,&dlfoe,&dlfoeout)  ;
    if(rc < 0)
    {
        // Error Calling MMC_DownloadFoE. Error in dlfoeout.usErrorID
        return ;
    }
    //
    // If we reached this line, the tftp was succesful. Poll the GMAS for results:
    while(TRUE)
    {
        Sleep(100) ;
        //
        // Check the FoE progress:
        MMC_GetFoEStatus(conn_hndl,&foestat);
        if(rc < 0)
        {
            // Error Calling MMC_GetFoEStatus. Error in foestat.usErrorID
            return ;
        }
        //
        // Check that the FoE started.
        if(foestat.ucFOEStarted)
        {
            rc = MMC_GetGMASOperationMode(conn_hndl,&pOpmode) ;
            if(rc < 0)
            {
                // Error Calling MMC_GetGMASOperationMode. Error in pOpmode.usErrorID
                return ;
            }
```

```
                // Print Remaining time - foestat.ucProgress
                //
                // Check Foe Download progress is over and GMAS back in operational mode.
                if ((foestat.ucProgress == 0) && (pOpmode.ucResult == 0))
                {
                     //
                     // Download over. Check to see if any drives failed.
                     for(i = 0 ; i < dlfoe.ucSlavesNum ; i++)
                     {
                          if(foestat.pstSlavesErrorID[i].sErrorID != 0)
                          {
                               // Error on one of the slaves. Print error:
                               // SlaveID: foestat.pstSlavesErrorID[i].usSlaveID  has error -
foestat.pstSlavesErrorID[i]
                          }
                     }
                     // Notify user to switch drives Off / On and then check the download status.
wait 5 sec's.
                     //
                     // please note - MAX 76 slaves can be read.
                     rc = MMC_GetEthercatCommStatistics(conn_hndl,&gcstat_In,&gcstat_Out) ;
                     if(rc < 0)
                     {
                          // Error Calling MMC_GetEthercatCommStatistics. Error in
gcstat_Out.usErrorID
                          return ;
                     }
                     //
                     // gcstat_Out.ucMasterState - Should be EcatStateO  . operational.
                     // Good idea to read number of slaves on bus - gcstat_Out.usNumOfSlaves
                     // gcstat_Out.ucMasterDiagnosticState - All bits should be 0, except for:
EcatMasterDiagnosticStateUpdated, EcatMasterDiagnosticStateDefaultDataWasSet bits.
                     //
                     for(i = 0 ; i < gcstat_In.ucSlavesNum ; i++)
                     {
                          // gcstat_Out.pucAxesState[i] - Should be EcatStateO.
                          // gcstat_Out.pucAxesDiagnosticState[i] - Should be 0.
                          if((gcstat_Out.pstSII_Content[i].ulVendorId == 0x9A) &&
(gcstat_Out.pstSII_Content[i].ulRevisionNo <= 0xFF))
                          {
                               //
                               // Drive stuck in no firmware state. Notify User. In This case the
InitCmdFail in diagnostics is not relevant.
                          }
                     }
                     return ;
                }
           }
      }
}

int OnConnectGetDiagnostics()
{
     int rc ;
     MMC_GET_GMASOP_MODE_OUT pOpmode ;

     MMC_GETCOMMSTATISTICSEX_IN gcstat_In ;
     MMC_GETCOMMSTATISTICSEX_OUT gcstat_Out ;
     int i ;
     //
     rc = MMC_GetGMASOperationMode(conn_hndl,&pOpmode) ;
     if(rc < 0)
     {
          // Error Calling MMC_GetGMASOperationMode. Error in pOpmode.usErrorID
          return ;
     }
     //
     // Check GMAS Operational state. If == 2, then in Download FOE state.
     if (pOpmode.ucResult == 2)
```

```
     {
          // GMAS in Download FoE state. We decided that a mesage will be shown to user that
the GMAS is in Download FoE.
     }

     rc = MMC_GetEthercatCommStatistics(conn_hndl,&gcstat_In,&gcstat_Out) ;
     if(rc < 0)
     {
          // Error Calling MMC_GetEthercatCommStatistics. Error in gcstat_Out.usErrorID
          return ;
     }
     //
     // gcstat_Out.ucMasterState - Should be EcatStateO. operational.
     // Good idea to read number of slaves on bus - gcstat_Out.usNumOfSlaves. Should be
identical to number of drives configured.
     // gcstat_Out.ucMasterDiagnosticState - All bits should be 0, except for:
EcatMasterDiagnosticStateUpdated, EcatMasterDiagnosticStateDefaultDataWasSet bits.
     //
     for(i = 0 ; i < gcstat_In.ucSlavesNum ; i++)
     {
          // gcstat_Out.pucAxesState[i] - Should be EcatStateO.
          //
          if((gcstat_Out.pstSII_Content[i].ulVendorId == 0x9A) &&
(gcstat_Out.pstSII_Content[i].ulRevisionNo <= 0xFF))
          {
               //
               // One of the Drives is stuck in no firmware state. Notify User to go to
diagnostics tab - NOT TO CONFIGURATOR.
               // In this case - the gcstat_Out.pucAxesDiagnosticState[i] InitCmd bit may be
set..
          }
          else
          {
               // pucAxesDiagnosticState[i] should be 0 !
          }

     }
}
```

10.3.16. MMC_GetEnquireFbStatus

Obtains the current state global parameter Receive FB status in EAS.

```
MMC_LIB_API int MMC_GetEnquireFbStatusCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETENQUIREFBSTATUS_IN* pInParam,
OUT MMC_GETENQUIREFBSTATUS_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETENQUIREFBSTATUS_IN** input data structure using the MMC_GetEnquireFbStatus function.

pOutParam

Points to the **MMC_GETENQUIREFBSTATUS_OUT** output structure receiving information, as a result of calling the MMC_GetEnquireFbStatus function.

Remarks

This function is only for IPC programming in C, and C++, and allows the user to control the release mode of the function blocks. By default it is read from the G-MAS resource file and should be set to False (0). If set to True (1) , the FBs will not be released from the FB queue until the user releases them implicitly. For EAS and any RPC programming, it will be set automatically to False (0) without user control.

In the IEC61131-3 program, the function parameter is also read from the G-MAS resource file and is automatically set to True (1). This is because the IEC61131-3 function blocks require and provide real data from the G-MAS and drives attached, during their motion, as they load and run each function block in turn. The function block will only be released from the queue when the G-MAS and drives perform that function block and move to the next function block.

Scope

All

MMC_GETENQUIREFBSTATUS_IN Structure

```
typedef struct mmc_getenquirefbstatus_in{
unsigned char ucDummy;
} MMC_GETENQUIREFBSTATUS_IN;
```

Parameters

dummy

Dummy character. Any +ve value accepted.

MMC_GETENQUIREFBSTATUS_OUT Structure

```
typedef struct mmc_getenquirefbstatus_out{
unsigned char ucCurrentStatus;
unsigned short usStatus;
short usErrorID;
} MMC_GETENQUIREFBSTATUS_OUT;
```

Parameters

ucCurrentStatus

The current status of the function block, with +ve character values of 0 - 255.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-16describes the function for MMC_GetEnquireFbStatus

Figure 10-16: MMC_GetEnquireFbStatus function

10.3.16.2. Function Block Code Example

10.3.17. MMC_GetAxisByName

Returns an axis index reference by its name.

```
MMC_LIB_API int MMC_GetAxisByNameCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXISBYNAME_IN* pInParam,
OUT MMC_AXISBYNAME_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_AXISBYNAME_IN** input data structure using the MMC_GetAxisByName function.

pOutParam

Points to the **MMC_AXISBYNAME_OUT** output structure receiving information, as a result of calling the MMC_GetAxisByName function.

Remarks

None

Scope

All

MMC_AXISBYNAME_IN Structure

```
typedef struct{
char cAxisName[NODE_NAME_MAX_LENGTH];
}MMC_AXISBYNAME_IN;
```

Parameters

cAxisName

Axis name. Any +ve character value.

[NODE_NAME_MAX_LENGTH] is an array of node maximum name lengths with values [1….n].

MMC_AXISBYNAME_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned short usAxisIdx;
}MMC_AXISBYNAME_OUT;
```

Parameters

usAxisIdx

Axis reference. Specific integer value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-17 describes the function block for MMC_GetAxisByName

Figure 10-17: MMC_GetAxisByName function block

10.3.17.2. Function Block Code Example

```
int rc;
MMC_AXISBYNAME_IN      stAxisByName_in;
MMC_AXISBYNAME_OUT     stAxisByName_out;
//
// Inserting the structure parameters:
strcpy(stAxisByName_in.cAxisName, "1st_Axis_name");
strcpy(stAxisByName_in.cAxisName, "2nd_Axis_name");
strcpy(stAxisByName_in.cAxisName, "3rd_Axis_name");
//
rc = MMC_GetAxisByNameCmd (hConn, &stAxisByName_in, &stAxisByName_out);
printf("Axis State[%ld] ErrId[%d]\n", (long int)stAxisByName_out.usAxisIdx,
(short)stAxisByName_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

10.3.18. MMC_GetGroupByName

Returns a group index reference by its name.

```
MMC_LIB_API int MMC_GetGroupByNameCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXISBYNAME_IN* pInParam,
OUT MMC_AXISBYNAME_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_AXISBYNAME_IN** input data structure using the MMC_GetGroupByName function.

pOutParam

Points to the **MMC_AXISBYNAME_OUT** output structure receiving information, as a result of calling the MMC_GetGroupByName function.

Remarks

None

Scope

All

MMC_AXISBYNAME_IN Structure

```
typedef struct{
char cAxisName[NODE_NAME_MAX_LENGTH];
}MMC_AXISBYNAME_IN;
```

Parameters

cAxisName

Axis name. Any +ve character value.

[NODE_NAME_MAX_LENGTH] is an array of node maximum name lengths with values [1….n].

MMC_AXISBYNAME_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned short usAxisIdx;
}MMC_AXISBYNAME_OUT;
```

Parameters

usAxisIdx

Axis index reference. Specific integer bitwise value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-18 describes the function block for MMC_GetGroupByName

Figure 10-18: MMC_GetGroupByName function block

10.3.18.2. Function Block Code Example

```
int rc;
MMC_AXISBYNAME_IN      stAxisByName_in;
MMC_AXISBYNAME_OUT     stAxisByName_out;
//
// Inserting the structure parameters:
strcpy(stAxisByName_in.cAxisName, "1st_Group_name");
strcpy(stAxisByName_in.cAxisName, "2nd_Group_name");
strcpy(stAxisByName_in.cAxisName, "3rd_Group_name");
//
rc = MMC_GetGroupByNameCmd (hConn, &stAxisByName_in, &stAxisByName_out);
printf("Group State[%ld] ErrId[%d]\n", (long int)stAxisByName_out.usAxisIdx,
(short)stAxisByName_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

10.3.19. MMC_GetGroupMembersInfo

Returns information about a specific group and its members.

```
MMC_LIB_API int MMC_GetGroupMembersInfo(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GETGROUPMEMBERSINFO_IN * pInParam,
OUT MMC_GETGROUPMEMBERSINFO_OUT * pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GETGROUPMEMBERSINFO_IN** input data structure using the MMC_GetGroupMembersInfo function.

pOutParam

Points to the **MMC_GETGROUPMEMBERSINFO_OUT** output structure receiving information, as a result of calling the MMC_GetGroupMembersInfo function.

Remarks

None

Scope

All

MMC_GETGROUPMEMBERSINFO_IN Structure

```
t typedef struct mmc_getgroupmembersinfo_in{
unsigned char ucDummy;
} MMC_GETGROUPMEMBERSINFO_IN;
```

Parameters

ucDummy

Any dummy value accepted.

MMC_GETGROUPMEMBERSINFO_OUT Structure

```
typedef struct mmc_getgroupmembersinfo_out{
char pAxesNames[NC_MAX_NUM_AXES_IN_NODE][NODE_NAME_MAX_LENGTH];
unsigned short pAxesReferences[NC_MAX_NUM_AXES_IN_NODE];
unsigned short pDeviceID[NC_MAX_NUM_AXES_IN_NODE];
unsigned short usStatus;
short usErrorID;
unsigned char ucNumOfAxes;
} MMC_GETGROUPMEMBERSINFO_OUT;
```

Parameters

pAxesNames[NC_MAX_NUM_AXES_IN_NODE][NODE_NAME_MAX_LENGTH]

Name recorded in the Resource file of the G-MAS. Value is a set of characters.

Array of axes names. The array size is equal to 16, as the maximum number of axes in a group.

[NODE_NAME_MAX_LENGTH] is the maximum name lengths = 80 characters.

pAxesReferences[NC_MAX_NUM_AXES_IN_NODE]

Reference to each axis handle in the group. Has values of any +ve number.

Array of axes references. The array size is equal to 16, as the maximum number of axes in a group.

pDeviceID[NC_MAX_NUM_AXES_IN_NODE]

Bus device ID. Has values of any +ve number.

Array of device IDs. The array size is equal to 16, as the maximum number of axes in a group.

ucNumOfAxes

The number of axes in the group. Any number set [0….16] is accepted.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-19 describes the function block for MMC_GetGroupMembersInfo

Figure 10-19: MMC_GetGroupMembersInfo function block

10.3.19.2. Function Block Code Example

```
// Declarate variables for connection
MMC_CONNECT_HNDL hConn;
MMC_IPC_CONNECTION_PARAM_STRUCT sInitConnectionInParam;
sInitConnectionInParam.uiParams = 1;
//
// Create connection (assume that there are not errors in this function)
MMC_IPCInitConnection(sInitConnectionInParam,NULL,&hConn);
//
//
MMC_AXISBYNAME_IN  sGroupByNameInParam;
MMC_AXISBYNAME_OUT sGroupByNameOutParam;
strcpy(sGroupByNameInParam.cAxisName,"v01");
//
// Get group reference (assume that there are not errors in this function)
MMC_GetGroupByNameCmd(hConn,&sGroupByNameInParam,&sGroupByNameOutParam);
//
// Create input and output structures
MMC_GETGROUPMEMBERSINFO_IN  sMembersInfoInParam;
MMC_GETGROUPMEMBERSINFO_OUT sMembersInfoOutParam;
//
// There are no neccessary inputs in the input structure (only dummy variable)
sMembersInfoInParam.ucDummy = 0;
//
// call GetGroupMembersInfo function (assume that there are not errors in this function)
MMC_GetGroupMembersInfo(hConn,sGroupByNameOutParam.usAxisIdx,&sMembersInfoInParam,&sMembersInf
oOutParam);
//
// Read the outputs
printf("The Vector [%s] has %d
axes.\n\n",sGroupByNameInParam.cAxisName,sMembersInfoOutParam.ucNumOfAxes);
for(int i = 0; i < sMembersInfoOutParam.ucNumOfAxes; i++)
{
// Print axes data
printf("Axis - [%s]\n",sMembersInfoOutParam.pAxesNames[i]);
printf("\tReference = [%d]\n",sMembersInfoOutParam.pAxesReferences[i]);
printf("\tDevice ID = [%d]\n\n",sMembersInfoOutParam.pDeviceID[i]);
}
//
```

10.3.19.3. Implementation Example

Run example in real system and get the following results. In order to obtain these results, this example should be run with a specific resource file:

The Vector v01 has 5 axes.

Axis - [b01]

Reference = [0]

Device ID = [1]

Axis - [b02]

Reference = [1]

Device ID = [2]

Axis - [b03]

Reference = [2]

Device ID = [3]

Axis - [b04]

Reference = [3]

Device ID = [4]

Axis - [b05]

Reference = [4]

Device ID = [5]

10.3.20. MMC_GetGMASOperationMode

Returns the current GMAS operation mode.

```
MMC_LIB_API int MMC_GetGMASOperationMode(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GET_GMASOP_MODE_IN* pInParam
OUT MMC_GET_GMASOP_MODE_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed - Not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GET_GMASOP_IN** input data structure using the MMC_GetGMASOperationMode function.

pOutParam

Points to the **MMC_GET_GMASOP_OUT** output structure receiving information, as a result of calling the MMC_GetGMASOperationMode function.

Remarks

None

Scope

All

MMC_GET_GMASOP_IN Structure

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

Parameters

dummy

Dummy input. Any +ve character value.

MMC_GET_GMASOP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned char ucResult;
}MMC_GET_GMASOP_MODE_OUT;
```

Parameters

ucResult

Returned answer to question “Is the EtherCAT Config mode operational?” Yes or No. Boolean acceptable values of 0 or 1 accepted.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-20 describes the function block for MMC_GetGMASOperationMode.

Figure 10-20: MMC_GetGMASOperationMode function block

10.3.20.2. Function Block Code Example

```
int rc;
MMC_GET_GMASOP_MODE_IN      stGetGMASOpMode_in;
MMC_GET_GMASOP_MODE_OUT     stGetGMASOpMode_out;
//
// Inserting the structure parameters:
stGetGMASOpMode_in.ucDummy      = 1;      // Dummy input
//
rc = MMC_GetGMASOperationMode (hConn, &stGetGMASOpMode_out);
printf("GMAS Operation State[%ld] ErrId[%d]\n", (long int)stGetGMASOpMode_out.ucResult,
(short)stGetGMASOpMode_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

10.3.21. MMC_GetStatusRegister

The purpose of the function is to provide usable information regarding the G-MAS and axes statuses.

```
int MMC_GetStatusRegisterCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GETSTATUSREGISTER_IN* pInParam,
OUT MMC_GETSTATUSREGISTER_OUT* pOutParam
);
```

Motion Mode	NC – Supported	Distributed – Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GETSTATUSREGISTER_IN** input data structure using the MMC_GetStatusRegister function.

pOutParam

Points to the **MMC_GETSTATUSREGISTER_OUT** output structure receiving information as a result of calling the MMC_GetStatusRegister function.

Remarks

None

Scope

All

MMC_GETSTATUSREGISTER_IN Structure

```
typedef struct mmc_getstatusregister_in{
unsigned char dummy;
} MMC_GETSTATUSREGISTER_IN;
```

Parameters

dummy

Any dummy value.

MMC_GETSTATUSREGISTER_OUT Structure

```
typedef struct mmc_getstatusregister_out{
unsigned int uiStatusRegister;
unsigned int uiMcsLimitRegister;
unsigned short usStatus;
short usErrorID;
unsigned char ucEndMotionReason
unsigned char cBuffer[32];
} MMC_GETSTATUSREGISTER_OUT;
```

Parameters

uiStatusRegister

This the Status Register, a 32 bit status register, with 10 lower bits related to the hardware/software limits feature. All other bits will be used in the future.

uiMcsLimitRegister

This is the MCS Limit Register, a 32 bit representation of the software limit status of all kinematic directions, 16 directions * 2 limits (High\Low) = 32

ucEndMotionReason

Inform reason for motion ending. +ve value char data.

cBuffer[32]

cBuffer is currently not in use, and this memory is allocated for future uses.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-21 describes the function block for MMC_GetStatusRegister

Figure 10-21: MMC_GetStatusRegister function block

10.3.21.2. Function Block Code Example

```
     CMMCConnection MyConnectionClass;
     MMC_CONNECT_HNDL ui_conn_hndl;
//
// Create Connection
     ui_conn_hndl = MyConnectionClass.ConnectIPC(0x7FFFFFFF,NULL);
//
// Initiate axis
     CMMCSingleAxis AxisA, AxisB;
     AxisA.InitAxisData("b01",ui_conn_hndl);
     AxisB.InitAxisData("b02",ui_conn_hndl);
//
     MMC_GETSTATUSREGISTER_IN StatusIN;
     MMC_GETSTATUSREGISTER_OUT StatusOUT;
     iRetval = MMC_GetStatusRegisterCmd(ui_conn_hndl,AxisA.GetRef(),&StatusIN,&StatusOUT);

          if(iRetval != NC_OK)
// Error handling
}
          else
{
// Status Register
          StatusOUT.uiStatusRegister;
// If the axis in SW Low limit the value will be 4 in decimal and 100 in binary

// MCS limit register relevant for group represents the status of MCS limits
          StatusOUT.uiMcsLimitRegister;

// Status
          StatusOUT.usStatus;

// Error ID
          StatusOUT.usErrorID;

// Future use
          StatusOUT.cBuffer[32];
}
```

10.3.22. MMC_GetResList

Returns the list of all resource files.

```
MMC_LIB_API int MMC_GetResListCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GET_RESLIST_IN* pInParam,
OUT MMC_GET_RESLIST_OUT* pOutParam);
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GET_RESLIST_IN** input data structure using the MMC_GetResList function.

pOutParam

Points to the **MMC_GET_RESLIST_OUT** output structure receiving information, as a result of calling the MMC_GetResList function.

Remarks

None

Scope

All

MMC_GET_RESLIST_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_GET_RESLIST_IN;
```

Parameters

dummy

Dummy input. Any +ve character value.

MMC_GET_RESLIST_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
char pResList[1024];
}MMC_GET_RESLIST_OUT;
```

Parameters

pResList[1024]

Resource file list. Any +ve characters limited to 1024 chars

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-22 describes the function block for MMC_GetResList

Figure 10-22: MMC_GetResList function block

10.3.22.2. Function Block Code Example

```
int rc;
MMC_GET_RESLIST_IN          stGetResList_in;
MMC_GET_RESLIST_OUT         stGetResList_out;
//
// Inserting the structure parameters:
stGetResList_in.dummy       = 1;      //Dummy input
//
rc = MMC_GetResListCmd (hConn, &stGetResList_in, &stGetResList_out);
printf("Resource List State[%ld] ErrId[%d]\n", (long int)stGetResList_out.pResList,
(short)stGetResList_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

10.3.23. MMC_GetResSnapshot

Save the resource configuration to temporary snapshot file.

```
MMC_LIB_API int MMC_GetResSnapshotCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_RESSNAPSHOT_IN* pInParam,
OUT MMC_RESSNAPSHOT_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_RESSNAPSHOT_IN** input data structure using the MMC_GetResSnapshot function.

pOutParam

Points to the **MMC_RESSNAPSHOT_OUT** output structure receiving information, as a result of calling the MMC_GetResSnapshot function.

Remarks

None

Scope

All

MMC_RESSNAPSHOT_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_RESSNAPSHOT_IN;
```

Parameters

dummy

Dummy input. Any +ve character value.

MMC_RESSNAPSHOT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_RESSNAPSHOT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-23 describes the function block for MMC_GetResSnapshot

Figure 10-23: MMC_GetResSnapshot function block

10.3.23.2. Function Block Code Example

```
int rc;
MMC_RESSNAPSHOT_IN          stResSnapshot_in;
MMC_RESSNAPSHOT_OUT         stResSnapshot_out;
//
// Inserting the structure parameters:
stResSnapshot_in.dummy = 1;     //Dummy input
//
rc = MMC_GetResSnapshotCmd (hConn, &stResSnapshot_in, &stResSnapshot_out);
printf("ErrId[%d]\n", (short)stResSnapshot_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

10.3.24. MMC_GetVersion

Obtains the G-MAS version in the output parameter.

```
MMC_LIB_API int MMC_GetVersionCmd(
IN MMC_CONNECT_HNDL hConn,
OUT MMC_GET_VER_OUT* sVersion
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

sVersion

Version data structure. Points to the **MMC_GET_VER_OUT** output structure receiving information, as a result of calling the MMC_GET_VER_OUT structure.

Remarks

This version reference consists of two parts:

- Firmware
- UBoot (bootloader which the G-MAS CPU reads from the flash upon startup)

The maximal length of the MMC_GetVersion output string is 120 characters.

Scope

All

MMC_GET_VER_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_GET_VER_IN;
```

Parameters

dummy

Any dummy value.

MMC_GET_VER_OUT Structure

```
typedef struct{
unsigned int uiUbootVer;
unsigned short usStatus;
short usErrorID;
char cFirst;
char cSecond;
char cThird;
char cFourth;
}MMC_GET_VER_OUT;
```

Parameters

uiUbootVer

U-Boot version. Any +ve integer.

cFirst

First byte of the G-MAS version. Any character.

cSecond

Second byte of the G-MAS version. Any character.

cThird

Third byte of the G-MAS version. Any character.

cFourth

Fourth byte of the G-MAS version. Any character.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-24 describes the function block for MMC_GetVersion

Figure 10-24: MMC_GetVersion function block

10.3.24.2. Function Block Code Example

```
int rc;
MMC_GET_VER_IN          stGetVer_in;
MMC_GET_VER_OUT         stGetVer_out;
//
// Inserting the structure parameters:
stGetVer_in.dummy = 1;      //Function block handle
//
rc = MMC_GetVersionCmd (hConn, &stGetVer_out);
printf("Version Status[%ld] ErrId[%d]\n", (long int)stGetVer_out.uiUbootVer,
(short)stGetVer_out.usErrorID);
printf("Version Status[%ld] [%ld] [%ld] [%ld]\n", (long int)stGetVer_out.cFirst, (long
int)stGetVer_out.cSecond, (long int)stGetVer_out.cThird, (long int)stGetVer_out.cFourth);
if (rc != 0)
{
    HandleError();
}
```

10.3.25. MMC_GetVersionEx

Obtains the G-MAS extended version in the output parameter.

```
MMC_LIB_API int MMC_GetVersionExCmd(
IN MMC_CONNECT_HNDL hConn,
OUT MMC_GET_VEREX_OUT* sVersion
);
```

Motion Mode NC – Not relevant Distributed – not relevant

Source GMAS\includes\MMC_general_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

sVersion

Version data structure. Points to the **MMC_GET_VEREX_OUT** output structure receiving information, as a result of calling the MMC_GET_VEREX_OUT structure.

Remarks

This version reference consists of two parts:

- Firmware
- UBoot (bootloader which the G-MAS CPU reads from the flash upon startup)

The maximal length of the MMC_GetVersion output string is 120 characters.

Scope

All

MMC_GET_VEREX_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_GET_VEREX_IN;
```

Parameters

dummy

Any dummy value.

MMC_GET_VEREX_OUT Structure

```
typedef struct{
unsigned short usStatus;                //Returned command status
short usErrorID;                        //Returned command error ID
char pcData[MAX_GETVERSION_CHARS];
} MMC_GET_VEREX_OUT;
```

Parameters

pcData[MAX_GETVERSION_CHARS]

Output string containing details about the GMAS firmware version, creation firmware date and time, uboot version and processor revision ID. Only necessary to print the string for convenience.

For example, output from a G-MAS with current firmware version: 1115 is:
creation date: Oct 18 2012, creation time: 16:53:29, uboot version: 9, revision ID: 0x10

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-25 describes the function block for MMC_GetVersionEx

Figure 10-25: MMC_GetVersionEx function block

10.3.25.2. Function Block Code Example

```
int rc;
MMC_GET_VER_IN          stGetVer_in;
MMC_GET_VER_OUT         stGetVer_out;
//
// Inserting the structure parameters:
stGetVer_in.dummy = 1;      //Function block handle
//
MMC_GET_VEREX_OUT GetVerOut;
MMC_CONNECT_HNDL ConnHndl ;

rc = MMC_GetVersionExCmd(ConnHndl,&GetVerOut);
if(rc != 0)
     {
     printf("MMC_GetVersionExCmd failed, error %d", GetVerOut.usErrorID);
     goto lbl_exit;
     }
printf("%s\n",GetVerOut.pcData);
```

10.3.26. MMC_InitConnection

Initiates connection to the G-MAS server.

```
MMC_LIB_API int MMC_InitConnection(
IN MMC_CONNECTION_TYPE eType,
IN MMC_CONNECTION_PARAM_STRUCT sConnParam,
IN MMC_CB_FUNC pCbFunc,
OUT MMC_CONNECT_HNDL* pHndl
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

eType

[IN] Connection type (IPC(inter process communication) or RPC(remote procedure calls)) with the following possible MMC_CONNECTION_TYPE enumerator values:

MMC_RPC_CONN_TYPE	RPC connection to MMC server
MMC_IPC_CONN_TYPE	IPC connection to MMC server

sConnParam

[IN] Connection parameters. (e.g. IP port, UDP port for callback)

uiTcpPort

TCP Port. Any +ve integer value.

uiCbUdpPort

UDP Port. Any +ve integer value.

ucIp[16]

IP address. 16-bit character.

pCbFunc

[IN] MMC_CB_FUNC pointer to UDP callback function using pCbFunc. No character value or short integer. The source for the parameter is:

GMAS\includes\MMC_definitions.h

pHndl

[OUT] Connection handle output using pHndl, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

Remarks

None

Scope

This function initiates an IPC or RPC connection to the G-MAS host.

Figure 10-26 describes the function block for MMC_InitConnection

Figure 10-26: MMC_InitConnection function block

10.3.26.1. Function Block Code Example

```
int rc;
MMC_CONNECTION_PARAM_STRUCT stsConnParam;
//
// Inserting the structure parameters:
strcpy ((char*)stsConnParam.ucIp, "255.0.110.0");      //IP address
stsConnParam.uiCbUdpPort    = 1520;                 //UDP Port
stsConnParam.uiTcpPort      = 1910;                 //TCP Port
eType[1]                    = MMC_IPC_CONN_TYPE;    //Connection type
pCbFunc                     = NULL;                 //Pointer to UDP callback function
//
rc = MMC_InitConnection ((MMC_CONNECTION_TYPE) eType, stsConnParam, (MMC_CB_FUNC) pCbFunc,
&hConn);
printf("Connection State[%ld]\n", (long int)(MMC_CONNECT_HNDL) pHndl);
if (rc != 0)
{
    HandleError();
}
```

10.3.27. MMC_IPCInitConnection

Initiates IPC connection to G-MAS server.

```
MMC_LIB_API int MMC_IPCInitConnection(
IN MMC_IPC_CONNECTION_PARAM_STRUCT sConnParam,
IN MMC_CB_FUNC pCbFunc,
OUT MMC_CONNECT_HNDL* pHndl
);
```

Motion Mode NC – Not relevant Distributed – not relevant

Source GMAS\includes\MMC_general_API.h

Function Parameters

sConnParam

[IN] Connection parameters. (e.g. IP port, UDP port for callback) as a function of the parameter MMC_IPC_CONNECTION_PARAM_STRUCT.

```
typedef struct{
unsigned int uiParams;
} MMC_IPC_CONNECTION_PARAM_STRUCT
```

Sub-function Parameters

uiParams

Currently: bit0 an explicit connection. All other connections will be closed. +ve valued integer.

uiTcpPort

TCP Port. Any +ve integer value.

uiCbUdpPort

UDP Port. Any +ve integer value.

ucIp[16]

IP address. 16-bit character.

pCbFunc

[IN] MMC_CB_FUNC pointer to UDP callback function using pCbFunc. No character value or short integer. The source for the parameter is:

GMAS\includes\MMC_definitions.h

pHndl

[OUT] Connection handle output using pHndl, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**.. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

Remarks

None

Scope

This function is specifically designed for IPC connections.

Figure 10-29 describes the function block for MMC_IPCInitConnection

Figure 10-27: MMC_IPCInitConnection function block

10.3.27.1. Function Block Code Example

```
int rc;
MMC_IPC_CONNECTION_PARAM_STRUCT     stsConnParam;
//
// Inserting the structure parameters:
stsConnParam.uiParams  = 1;      //IP address
pCbFunc                = NULL;   //Pointer to UDP callback function
//
rc = MMC_IPCInitConnection (stsConnParam, (MMC_CB_FUNC) pCbFunc, &hConn);
printf("Connection State[%ld]\n", (long int)(MMC_CONNECT_HNDL) pHndl);
if (rc != 0)
{
    HandleError();
}
```

10.3.28. MMC_LoadParam

Loads the axis, group, and global parameters from the xml file at the location:

//mnt/jffs/MMC/config/parameters/MMCParameters.xml

to the G-MAS

```
MMC_LIB_API int MMC_LoadParamCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_LOADPARAM_IN* pInParam,
OUT MMC_LOADPARAM_OUT* pOutParam
);
```

Motion Mode	NC - Not relevant	Distributed - Not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_LOADPARAM_IN** input data structure using the MMC_LoadParam function.

pOutParam

Points to the **MMC_LOADPARAM_OUT** output structure receiving information, as a result of calling the MMC_LoadParam function.

Remarks

None

Scope

All

MMC_LOADPARAM_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_LOADPARAMAXIS_IN;
```

Parameters

dummy

Any +ve dummy characters

MMC_LOADPARAM_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_LOADPARAMGLOBAL_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-28 describes the function block for MMC_LoadParam

Figure 10-28: MMC_LoadParam function block

10.3.28.2. Function Block Code Example

10.3.29. MMC_RpcInitConnection

Initiates RPC connection to G-MAS server.

```
MMC_LIB_API int MMC_RpcInitConnection(
IN MMC_CONNECTION_TYPE eType,
IN MMC_CONNECTION_PARAM_STRUCT sConnParam,
IN MMC_CB_FUNC pCbFunc,
IN char* cpHostIPAddr,
OUT MMC_CONNECT_HNDL* pHndl
);
```

Motion Mode NC – Not relevant Distributed – not relevant

Source GMAS\includes\MMC_general_API.h

Function Parameters

eType

[IN] Connection type (IPC(inter process communication) or RPC(remote procedure calls)) with the following possible MMC_CONNECTION_TYPE enumerator values:

MMC_RPC_CONN_TYPE	RPC connection to MMC server
MMC_IPC_CONN_TYPE	IPC connection to MMC server

sConnParam

[IN] Connection parameters. (e.g. IP port, UDP port for callback)

uiTcpPort

TCP Port. Any +ve integer value.

uiCbUdpPort

UDP Port. Any +ve integer value.

ucIp[16]

IP address. 16-bit character.

pCbFunc

[IN] MMC_CB_FUNC pointer to UDP callback function using pCbFunc. No character value or short integer. The source for the parameter is:

GMAS\includes\MMC_definitions.h

cpHostIPAddr

[IN] Host IP Address for multiple NIC support. Values accepted are the IP address characters. The source for the parameter is:

GMAS\includes\MMC_definitions.h

pHndl

[OUT] Connection handle output using pHndl, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions.

Returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

Remarks

None

Scope

This function is specifically designed for RPC connections, as it includes the parameter *cpHostIPAddr*, the host IP address for NIC support.

Figure 10-29 describes the function block for MMC_RpcInitConnection

Figure 10-29: MMC_RpcInitConnection function block

10.3.29.1. Function Block Code Example

```
int rc;
MMC_CONNECTION_PARAM_STRUCT stsConnParam;
//
// Inserting the structure parameters:
strcpy ((char*)stsConnParam.ucIp, "255.0.110.0");      //IP address
stsConnParam.uiCbUdpPort    = 1520;                    //UDP Port
stsConnParam.uiTcpPort      = 1910;                    //TCP Port
eType[1]                    = MMC_IPC_CONN_TYPE;       //Connection type
pCbFunc                     = NULL;                    //Pointer to UDP callback function
strcpy ((char*)cpHostIPAddr, "97.110.110.1");          //Host IP Address
//
rc = MMC_RpcInitConnection ((MMC_CONNECTION_TYPE) eType, stsConnParam, (MMC_CB_FUNC) pCbFunc,
cpHostIPAddr, &hConn);
printf("Connection State[%ld]\n", (long int)(MMC_CONNECT_HNDL) pHndl);
if (rc != 0)
{
    HandleError();
}
```

10.3.30. MMC_RpcInitConnectionEx

Initiates RPC connection to G-MAS server.

```
MMC_LIB_API int MMC_RpcInitConnectionEx(
IN MMC_CONNECTION_TYPE eType,
IN MMC_CONNECTION_PARAM_STRUCT sConnParam,
IN MMC_MB_CLBK pCbFunc,
IN char* cpHostIPAddr,
OUT MMC_CONNECT_HNDL* pHndl
);
```

Motion Mode NC – Not relevant Distributed – not relevant

Source GMAS\includes\MMC_general_API.h

Function Parameters

eType

[IN] Connection type (IPC(inter process communication) or RPC(remote procedure calls)) with the following possible MMC_CONNECTION_TYPE enumerator values:

MMC_RPC_CONN_TYPE	RPC connection to MMC server
MMC_IPC_CONN_TYPE	IPC connection to MMC server

sConnParam

[IN] Connection parameters. (e.g. IP port, UDP port for callback)

uiTcpPort

TCP Port. Any +ve integer value.

uiCbUdpPort

UDP Port. Any +ve integer value.

ucIp[16]

IP address. 16-bit character.

pCbFunc

[IN] MMC_MB_CLBK pointer to UDP callback function using pCbFunc. No character value or short integer. The source for the parameter is:

GMAS\includes\MMC_definitions.h

cpHostIPAddr

[IN] Host IP Address for multiple NIC support. Values accepted are the IP address characters. The source for the parameter is:

GMAS\includes\MMC_definitions.h

pHndl

[OUT] Connection handle output using pHndl, where MMC_CONNECT_HNDL is the

Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions.

Returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

Remarks

None

Scope

This function is specifically designed for RPC connections, as it includes the parameter *cpHostIPAddr*, the host IP address for NIC support.

Figure 10-29 describes the function block for MMC_RpcInitConnection

Figure 10-30: MMC_RpcInitConnection function block

10.3.30.1. Function Block Code Example

```
int rc;
MMC_CONNECTION_PARAM_STRUCT stsConnParam;
//
// Inserting the structure parameters:
strcpy ((char*)stsConnParam.ucIp, "255.0.110.0");      //IP address
stsConnParam.uiCbUdpPort    = 1520;                    //UDP Port
stsConnParam.uiTcpPort      = 1910;                    //TCP Port
eType[1]                    = MMC_IPC_CONN_TYPE;       //Connection type
pCbFunc                     = NULL;                    //Pointer to UDP callback function
strcpy ((char*)cpHostIPAddr, "97.110.110.1");          //Host IP Address
//
rc = MMC_RpcInitConnection ((MMC_CONNECTION_TYPE) eType, stsConnParam, (MMC_CB_FUNC) pCbFunc,
cpHostIPAddr, &hConn);
printf("Connection State[%ld]\n", (long int)(MMC_CONNECT_HNDL) pHndl);
if (rc != 0)
{
    HandleError();
}
```

10.3.31. MMC_ResetMultiAxisControl

Internal reset of the G-MAS multi-axis control. Allows the G-MAS's CPU to reset

```
MMC_LIB_API int MMC_ResetMultiAxisControl(
IN MMC_CONNECT_HNDL hConn,
IN MMC_EXIT_APP_IN *pInParam,
OUT MMC_EXIT_APP_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_EXIT_APP_IN** input data structure using the MMC_ResetMultiAxisControl function.

pOutParam

Points to the **MMC_EXIT_APP_OUT** output structure receiving information, as a result of calling the MMC_ResetMultiAxisControl function.

Remarks

This function closes MultiAxisControl by closing connection,shared memory, and nc_drv

Scope

All

MMC_EXIT_APP_IN Structure

```
typedef struct{
unsigned char ucEnable;
}MMC_EXIT_APP_IN;
```

Parameters

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_EXIT_APP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_EXIT_APP_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-31 describes the function block for MMC_ResetMultiAxisControl

Figure 10-31: MMC_ResetMultiAxisControl function block

10.3.31.2. Function Block Code Example

```
int rc;
MMC_EXIT_APP_IN        stExitApp_in;
MMC_EXIT_APP_OUT       stExitApp_out;
//
// Inserting the structure parameters:
stExitApp_in.ucEnable  = 1;          // Enabled
//
rc = MMC_ResetMultiAxisControl (hConn, &stExitApp_in, &stExitApp_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.32. MMC_ResExportFile

Copies a requested file from the G-MAS to the host via TFTP.

```
MMC_LIB_API int MMC_ResExportFileCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_RESEXPORTFILE_IN* pInParam,
OUT MMC_RESEXPORTFILE_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_RESEXPORTFILE_IN** input data structure using the MMC_ResExportFile function.

pOutParam

Points to the **MMC_RESEXPORTFILE_OUT** output structure receiving information, as a result of calling the MMC_ResExportFile function.

Remarks

The User file that can be downloaded is located in the following directory:

/mnt/jffs/user

Scope

All

MMC_RESEXPORTFILE_IN Structure

```
typedef struct{
char pFName[33];
char pServer[17];
char pFilePath[101];
char ucDownloadType;
}MMC_RESEXPORTFILE_IN;
```

Parameters

pFName[33]

Resource File Name. Any +ve character value with a limit of 33 characters

pServer[17]

Server name. Any +ve character value with a limit of 17 characters.

pFilePath[101]

Server file path. Any +ve character value with a limit of 101 characters.

ucDownloadType

File type to be copied. The following IDs are used to characterize the enumerator parameter:

ID	Enumerator value explanation
0	Parameter file
1	Resource file
2	Snapshot resource file
3	EtherCAT configuration file
4	Personality file
5	User file

MMC_RESEXPORTFILE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_RESEXPORTFILE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block.

Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-32 describes the function block for MMC_ResExportFile

Figure 10-32: MMC_ResExportFile function block

10.3.32.2. Function Block Code Example

```
int rc;
MMC_RESEXPORTFILE_IN        stResExportFile_in;
MMC_RESEXPORTFILE_OUT       stResExportFile_out;
//
// Inserting the structure parameters:
strcpy(stResExportFile_in.pFName,"Resource_File_Name");        //Resource File Name
strcpy(stResExportFile_in.pServer,"ServerName");               //Server Name
strcpy(stResExportFile_in.pFilePath,"Server_File_Path_Name");  //Server file path
stResExportFile_in.ucDownloadType   = 1;                       //Download type
//
rc = MMC_ResExportFileCmd (hConn, &stResExportFile_in, &stResExportFile_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.33. MMC_ResImportFile

Copies a requested file from host to the G-MAS via TFTP.

```
MC_LIB_API int MMC_ResImportFileCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_RESIMPORTFILE_IN* pInParam,
OUT MMC_RESIMPORTFILE_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_RESIMPORTFILE_IN** input data structure using the MMC_ResImportFile function.

pOutParam

Points to the **MMC_RESIMPORTFILE_OUT** output structure receiving information, as a result of calling the MMC_ResImportFile function.

Remarks

The User file that can be uploaded will be copied to the following directory:

/mnt/jffs/user

Scope

All

MMC_RESIMPORTFILE_IN Structure

```
typedef struct{
char pFName[33];
char pServer[17];
char ucDownloadType;
}MMC_RESIMPORTFILE_IN;
```

Parameters

pFName[33]

Resource File Name. Any +ve character value with a limit of 33 characters

pServer[17]

Server name. Any +ve character value with a limit of 17 characters.

ucDownloadType

File type to be copied. The following IDs are used to characterize the enumerator parameter:

ID	Enumerator value explanation
0	Parameter file
1	Resource file
2	Snapshot resource file
3	EtherCAT configuration file
4	Personality file. Not possible to import the Personality file from the host. Will produce an error if performed
5	User file

MMC_RESIMPORTFILE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_RESIMPORTFILE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-33 describes the function block for MMC_ResImportFile

Figure 10-33: MMC_ResImportFile function block

10.3.33.2. Function Block Code Example

```
int rc;
MMC_RESIMPORTFILE_IN        stResImportFile_in;
MMC_RESIMPORTFILE_OUT       stResImportFile_out;
//
// Inserting the structure parameters:
strcpy(stResImportFile_in.pFName, "Resource_File_Name");            //Resource File Name
strcpy(stResImportFile_in.pServer, "ServerName");                   //Server Name
stResImportFile_in.ucDownloadType   = 1;                            //Download type
//
rc = MMC_ResImportFileCmd (hConn, &stResImportFile_in, &stResImportFile_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.34. MMC_SaveParam

Save and/or update axes, group, and global parameters from the G-MAS to a file at:

//mnt/jffs/MMC/config/parameters/MMCParameters.xml

```
MMC_LIB_API int MMC_SaveParamCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SAVEPARAM_IN* pInParam,
OUT MMC_SAVEPARAM_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed - Not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SAVEPARAM_IN** input data structure using the MMC_SaveParam function.

pOutParam

Points to the **MMC_SAVEPARAM_OUT** output structure receiving information, as a result of calling the MMC_SaveParam function.

Remarks

None

Scope

All

MMC_SAVEPARAM_IN Structure

```
typedef struct{
unsigned char dummy;
} MMC_SAVEPARAMAXIS_IN;
```

Parameters

dummy

Dummy input. Any +ve character value.

MMC_SAVEPARAM_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
} MMC_SAVEPARAMAXIS_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-34 describes the function block for MMC_SaveParam.

Figure 10-34: MMC_SaveParam function block

10.3.34.2. Function Block Code Example

```
int rc;
MMC_SAVEPARAM_IN       stSaveParam_in;
MMC_SAVEPARAM_OUT      stSaveParam_out;
//
// Inserting the structure parameters:
stSaveParam_in.dummy       = 1;     // Dummy input
//
rc = MMC_SaveParamCmd (hConn, &stSaveParam_in, &stSaveParam_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.35. MMC_SetEnquireFbStatus

Sets the state global parameter Receive FB status in EAS.

```
MMC_LIB_API int MMC_SetEnquireFbStatusCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SETENQUIREFBSTATUS_IN* pInParam,
OUT MMC_SETENQUIREFBSTATUS_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SETENQUIREFBSTATUS_IN** input data structure using the MMC_SetEnquireFbStatus function.

pOutParam

Points to the **MMC_SETENQUIREFBSTATUS_OUT** output structure receiving information, as a result of calling the MMC_SetEnquireFbStatus function.

Remarks

This function is only for IPC programming in C, and C++, and allows the user to control the release mode of the function blocks. By default it is read from the G-MAS resource file and should be set to False (0). If set to True (1) , the FBs will not be released from the FB queue until the user releases them implicitly. For EAS and any RPC programming, it will be set automatically to False (0) without user control.

In the IEC61131-3 program, the function parameter is also read from the G-MAS resource file and is automatically set to True (1). This is because the IEC61131-3 function blocks require and provide real data from the G-MAS and drives attached, during their motion, as they load and run each function block in turn. The function block will only be released from the queue when the G-MAS and drives perform that function block and move to the next function block.

Scope

All

MMC_SETENQUIREFBSTATUS_IN Structure

```
typedef struct mmc_setenquirefbstatus_in{
unsigned char ucStatus;
} MMC_SETENQUIREFBSTATUS_IN;
```

Parameters

eStatus

The status of the function block, with +ve character values of 0 - 255.

MMC_SETENQUIREFBSTATUS_OUT Structure

```
typedef struct mmc_setenquirefbstatus_out{
unsigned short usStatus;
short usErrorID;
} MMC_SETENQUIREFBSTATUS_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-35 describes the function for MMC_SetEnquireFbStatus

Figure 10-35: MMC_SetEnquireFbStatus function

10.3.35.2. Function Block Code Example

10.3.36. MMC_SetDefaultParameters

Sets the G-MAS default manufacturer parameters to a specific Axis, or Group in the G-MAS.

```
MMC_LIB_API int MMC_SetDefaultParametersCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_SETDEFAULTPARAMETERS_IN* pInParam,
OUT MMC_SETDEFAULTPARAMETERS_OUT* pOutParam
);
```

Motion Mode	NC - Not relevant	Distributed - Not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_SETDEFAULTPARAMETERS_IN** input data structure using the MMC_SetDefaultParameters function.

pOutParam

Points to the **MMC_SETDEFAULTPARAMETERS_OUT** output structure receiving information, as a result of calling the MMC_SetDefaultParameters function.

Remarks

Resets the default manufacturer parameters to the specific axis/group in the G-MAS.

Scope

All

MMC_SETDEFAULTPARAMETERS_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_SETDEFAULTPARAMETERS_IN;
```

Parameters

dummy

Any dummy value

MMC_SETDEFAULTPARAMETERS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SETDEFAULTPARAMETERS_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 10-36 describes the function block for MMC_SetDefaultParameters

Figure 10-36: MMC_SetDefaultParameters function block

10.3.36.2. Function Block Code Example

10.3.37. MMC_SetDefaultParametersGlobal

Sets the G-MAS default manufacturer global parameters in the G-MAS.

```
MMC_LIB_API int MMC_SetDefaultParametersGlobalCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SETDEFAULTPARAMETERSGLOBAL_IN* pInParam,
OUT MMC_SETDEFAULTPARAMETERSGLOBAL_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed - Not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_RESIMPORTFILE_IN** input data structure using the MMC_SetDefaultParametersGlobal function.

pOutParam

Points to the **MMC_RESIMPORTFILE_OUT** output structure receiving information, as a result of calling the MMC_SetDefaultParametersGlobal function.

Remarks

Resets the default manufacturer global parameters to the G-MAS.

Scope

All

MMC_SETDEFAULTPARAMETERSGLOBAL_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_SETDEFAULTPARAMETERSGLOBAL_IN;
```

Parameters

dummy

Any dummy value.

MMC_SETDEFAULTPARAMETERSGLOBAL_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SETDEFAULTPARAMETERSGLOBAL_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-37 describes the function block for MMC_SetDefaultParametersGlobal

Figure 10-37: MMC_SetDefaultParametersGlobal function block

10.3.37.2. Function Block Code Example

10.3.38. MMC_SetIsToLoadGlobalParams

Defines a flag whether to load or not, the global parameters, when updating the global parameters from a file to the G-MAS.

```
MMC_LIB_API int MMC_SetIsToLoadGlobalParamsCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SETISTOLOADGLOBALPARAMS_IN* pInParam,
OUT MMC_SETISTOLOADGLOBALPARAMS_OUT* pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SETISTOLOADGLOBALPARAMS_IN** input data structure using the MMC_SetIsToLoadGlobalParams function.

pOutParam

Points to the **MMC_SETISTOLOADGLOBALPARAMS_OUT** output structure receiving information, as a result of calling the MMC_SetIsToLoadGlobalParams function.

Remarks

In general, there are at two instances when updating the global parameters from a file to the G-MAS:

- When using MMC_LoadParams
- After restarting (Power On or software restart) the G-MAS

Scope

All

MMC_SETISTOLOADGLOBALPARAMS_IN Structure

```
typedef struct{
unsigned char ucValue;
}MMC_SETISTOLOADGLOBALPARAMS_IN;
```

Parameters

ucValue

The function recieves either 0 (not required to load the set global parameters) or 1 (required to load the set global parameters). +ve integer value

MMC_SETISTOLOADGLOBALPARAMS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SETISTOLOADGLOBALPARAMS_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-38 describes the function block for MMC_SetIsToLoadGlobalParams

Figure 10-38: MMC_SetIsToLoadGlobalParams function block

10.3.38.2. Function Block Code Example

10.3.39. MMC_ShowNodeStat

Displays the debug information for an Axis/Group.

```
MMC_LIB_API int MMC_ShowNodeStatCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_SHOWNODESTAT_IN* pInParam,
OUT MMC_SHOWNODESTAT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_SHOWNODESTAT_IN** input data structure using the MMC_ShowNodeStat function.

pOutParam

Points to the **MMC_SHOWNODESTAT_OUT** output structure receiving information as a result of calling the MMC_ShowNodeStat function.

Remarks

None

Scope

All

MMC_SHOWNODESTAT_IN Structure

```
typedef struct{
unsigned int uiHndl;
}MMC_SHOWNODESTAT_IN;
```

Parameters

uiHndl

Requested function block handle. Integer with any +ve value

MMC_SHOWNODESTAT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SHOWNODESTAT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 10-39 describes the function block for MMC_ShowNodeStat

Figure 10-39: MMC_ShowNodeStat function block

10.3.39.2. Function Block Code Example

```
int rc;
MMC_SHOWNODESTAT_IN        stShowNodeStat_in;
MMC_SHOWNODESTAT_OUT       stShowNodeStat_out;
//
// Inserting the structure parameters:
stShowNodeStat_in.uiHndl   = 2;     // Requested function block handle
//
rc = MMC_ShowNodeStatCmd (hConn, iAxisRef, &stShowNodeStat_in, &stShowNodeStat_out);
if (rc != 0)
{
    HandleError();
}
```

10.3.40. MMC_GetActiveAxesNum

Displays the number of active axes attached and managed by the G-MAS.

```
MMC_LIB_API int MMC_GetActiveAxesNum(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETACTIVEAXESNUM_IN* pInParam,
OUT MMC_GETACTIVEAXESNUM_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GETACTIVEAXESNUM_IN** input data structure using the MMC_GetActiveAxesNum function.

pOutParam

Points to the **MMC_GETACTIVEAXESNUM_OUT** output structure receiving information as a result of calling the MMC_GetActiveAxesNum function.

Remarks

his function will provide this basic information without opening the G-MAS Personality file.

Scope

All

MMC_GETACTIVEAXESNUM_IN Structure

```
typedef struct {
unsigned char dummy;
}MMC_GETACTIVEAXESNUM_IN;
```

Parameters

dummy

Any dummy value.

MMC_GETACTIVEAXESNUM_OUT Structure

```
typedef struct {
int iActiveAxesNum;
unsigned short usStatus;
short usErrorID;
}MMC_GETACTIVEAXESNUM_OUT;
```

Parameters

iActiveAxesNum

Provides the number of active axes. Any +ve integer value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 10-40 describes the function block for MMC_GetActiveAxesNum

Figure 10-40: MMC_GetActiveAxesNum function block

10.3.40.2. Function Block Code Example

10.3.41. MMC_ToggleConsoleOutput

Toggles the Console output. This function is not available at this moment.

```
MMC_LIB_API int MMC_ToggleConsoleOutputCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_TOGGLECONSOLEOUTPUT_IN* pInParam,
OUT MMC_TOGGLECONSOLEOUTPUT_OUT* pOutParam
);
```

Motion Mode	NC – Supported?	Distributed – Supported?
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_TOGGLECONSOLEOUTPUT_IN** input data structure using the MMC_ToggleConsoleOutput function.

pOutParam

Points to the **MMC_TOGGLECONSOLEOUTPUT_OUT** output structure receiving information as a result of calling the MMC_ToggleConsoleOutput function.

Remarks

None

Scope

All

MMC_TOGGLECONSOLEOUTPUT_IN Structure

```
typedef struct mmc_toggleconsoleoutput_in{
unsigned char ucEnable;
} MMC_TOGGLECONSOLEOUTPUT_IN;
```

Parameters

ucEnable

This parameter is not in use and is no longer relevant. Dummy parameter.

MMC_TOGGLECONSOLEOUTPUT_OUT Structure

```
typedef struct mmc_toggleconsoleoutput_out{
short usErrorID;
} MMC_TOGGLECONSOLEOUTPUT_OUT;
```

Parameters

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 10-41 describes the function block for MMC_ToggleConsoleOutput

Figure 10-41: MMC_ToggleConsoleOutput function block

10.3.41.2. Function Block Code Example

10.3.42. MMC_GetCyclesCounter

Obtains the G-MAS cycles counter value

```
MMC_LIB_API int MMC_GetCyclesCounterCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETCYCLESCOUNTER_IN* pInParam,
OUT MMC_GETCYCLESCOUNTER_OUT* pOutParam
);
```

Motion Mode	NC – Supported	Distributed – Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETCYCLESCOUNTER_IN** input data structure using the MMC_GetCyclesCounter function.

pOutParam

Points to the **MMC_GETCYCLESCOUNTER_OUT** output structure receiving information as a result of calling the MMC_GetCyclesCounter function.

Remarks

None

Scope

All

MMC_GETCYCLESCOUNTER_IN Structure

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

Parameters

ucdummy

Any dummy value.

MMC_GETCYCLESCOUNTER_OUT Structure

```
typedef struct MMC_GETCYCLESCOUNTER_OUT{
unsigned long ulCyclesCounter;
unsigned short usStatus;
short usErrorID;
} MMC_GETCYCLESCOUNTER_OUT;
```

Parameters

ulCyclesCounter

Value of the cycles counter. Any +ve value acceptable.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 10-42 describes the function block for MMC_GetCyclesCounter

Figure 10-42: MMC_GetCyclesCounter function block

10.3.42.2. Function Block Code Example

10.3.43. MMC_WriteGroupOfParameters

This function write a group of parameters to the G-MAS.

```
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
);
```

Motion Mode	NC – Supported	Distributed – Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEGROUPOFPARAMETERS_IN** input data structure using the MMC_WriteGroupOfParameters function.

pOutParam

Points to the **MMC_WRITEGROUPOFPARAMETERS_OUT** output structure receiving information as a result of calling the MMC_WriteGroupOfParameters function.

Remarks

None

Scope

All

MMC_WRITEGROUPOFPARAMETERS_IN Structure

```
typedef struct mmc_writegroupofparameters_in{
MMC_WRITEGROUPOFPARAMETERSMEMBER sParameters[GROUP_OF_PARAMETERS_MAXIMUM_SIZE];
MC_EXECUTION_MODE eExecutionMode;
unsigned char ucNumberOfParameters;
unsigned char ucMode;
unsigned char ucExecute;
} MMC_WRITEGROUPOFPARAMETERS_IN;
```

Parameters

MMC_WRITEGROUPOFPARAMETERSMEMBER

```
typedef struct mmc_writegroupofparametersmember{
double dbValue;
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterIndex;
unsigned short usAxisRef;
unsigned short usPadding1;
unsigned short usPadding2;
unsigned short usPadding3;
}MMC_WRITEGROUPOFPARAMETERSMEMBER;
```

MMC_PARAMETER_LIST_ENUM eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterIndex

The parameter index, a +ve integer value

usAxisRef

Axis reference. Any +ve bitwise integer.

UsPadding1

Alignment padding of data. This parameter is not in use at this time.

UsPadding2

Alignment padding of data. This parameter is not in use at this time.

usPadding3

Alignment padding of data. This parameter is not in use at this time.

sParameters[GROUP_OF_PARAMETERS_MAXIMUM_SIZE]

The parameters of the group to be read within the array.

Array with the maximum value of GROUP_OF_PARAMETERS_MAXIMUM_SIZE is 5

eExecutionMode

Execution mode enumerator defining whether the execution is immediate or queued, with the following values:

eMMC_EXECUTION_MODE_IMMEDIATE = 0,

eMMC_EXECUTION_MODE_QUEUED

ucNumberOfParameters

The number of parameters in the array between 1 to 5, with 5 as a maximum number.

ucMode

RELATIVE =True, ABSOLUTE = False (Default)

RELATIVE means that Position is added to the actual position value of the axis at the time of execution. This results in a recalibration by a specified distance. ABSOLUTE means that the actual position value of the axis is set to the value specified in the Position parameter.

Values accepted are Boolean, TRUE/FALSE.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_WRITEGROUPOFPARAMETERS_OUT Structure

```
typedef struct mmc_writegroupofparameters_out{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
unsigned char ucProblematicEntry;
} MMC_WRITEGROUPOFPARAMETERS_OUT;
```

Parameters

uiHndl

Returned function block handle. Any +ve value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

ucProblematicEntry

The parameter entry which is faulty. +ve char value.

Figure 10-43 describes the function for MMC_WriteGroupOfParameters

Figure 10-43: MMC_WriteGroupOfParameters function

10.3.43.2. Function Block Code Example

10.3.44. MMC_ReadGroupOfParameters

This function retreive a group of parameters to the user.

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

Motion Mode	NC – Supported	Distributed – Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READGROUPOFPARAMETERS_IN** input data structure using the MMC_ReadGroupOfParameters function.

pOutParam

Points to the **MMC_READGROUPOFPARAMETERS_OUT** output structure receiving information as a result of calling the MMC_ReadGroupOfParameters function.

Remarks

None

Scope

All

MMC_READGROUPOFPARAMETERS_IN Structure

```
typedef struct mmc_readgroupofparameters_in{
MMC_READGROUPOFPARAMETERSMEMBER sParameters[GROUP_OF_PARAMETERS_MAXIMUM_SIZE];
unsigned char ucNumberOfParameters;
} MMC_READGROUPOFPARAMETERS_IN;
```

Parameters

MMC_READGROUPOFPARAMETERSMEMBER

```
typedef struct mmc_readgroupofparametersmember{
MMC_PARAMETER_LIST_ENUM eParameterNumber;
int iParameterIndex;
unsigned short usAxisRef;
unsigned short usPadding;
}MMC_READGROUPOFPARAMETERSMEMBER;
```

MMC_PARAMETER_LIST_ENUM eParameterNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3.2 Parameters Tables** for the appropriate integer parameter to be used as enumerator.

iParameterIndex

Array parameter index for arrayed parammeters only. Integer values.

usAxisRef

Axis reference. Any +ve bitwise integer.

usPadding

Alignment padding of data. This parameter is not in use at this time.

sParameters[GROUP_OF_PARAMETERS_MAXIMUM_SIZE]

The parameters of the group to be read within the array.

Array with the maximum value of GROUP_OF_PARAMETERS_MAXIMUM_SIZE is 5

ucNumberOfParameters

The total number of parameters in the group to be read. +ve character value.

MMC_READGROUPOFPARAMETERS_OUT Structure

```
typedef struct mmc_readgroupofparameters_out{
double dbValue[GROUP_OF_PARAMETERS_MAXIMUM_SIZE];
unsigned short usStatus;
short usErrorID;
unsigned char ucProblematicEntry;
} MMC_READGROUPOFPARAMETERS_OUT;
```

Parameters

dbValue[GROUP_OF_PARAMETERS_MAXIMUM_SIZE]

Value of the parameter requested within the group.

Array with the maximum value of GROUP_OF_PARAMETERS_MAXIMUM_SIZE is 5

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

ucProblematicEntry

The G-MAS returns feedback on a problematic parameter, or which is incorrectly set in the G-MAS. Unlimited character value allowed.

Figure 10-44 describes the function for MMC_ReadGroupOfParameters

Figure 10-44: MMC_ReadGroupOfParameters function

10.3.44.2. Function Block Code Example

10.3.45. MMC_GetVerPath

Reads the G-MAS firmware flash version path from the G-MAS itself. This is set during a TFT firmware download from the host server.

When it becomes necessary to use this function block, please contact Elmo for further information.

10.3.46. MMC_DownloadVersion

Performs a download of new firmware version to the G-MAS. When it becomes necessary to use this function block, please contact Elmo for further information.

10.3.47. MMC_ReadDownloadVersionStatus

Returns the status of the last downloaded version.

When it becomes necessary to use this function block, please contact Elmo for further information.

10.3.48. MMC_SetVerPath

As part of the download firmware functions, sets the G-MAS firmware flash version path from the G-MAS itself. This is set during a TFT firmware download from the host server.

When it becomes necessary to use this function block, please contact Elmo for further information.

Chapter 11: Connectivity and Configuration

The following sections describe the connectivity and configuration function blocks utilized in the G-MAS to communicate with a host and other devices described in the section **Chapter** 2: **G-MAS Overview**.

11.1. Network Function Blocks

The following function blocks are utilized in the network communications to, and as part of the G-MAS. The network node host system is defined by its host IP addresses, which are communicated to the G-MAS FLASH. The following network function blocks are described, with the exclusion of MMC_NodesInfo, which is a structure:

Network
MMC_CloseUdpChannel
MMC_GetDefGateway
MMC_GetIpAddr
MMC_GetIpMask
MMC_GetServerIp
MMC_NetworkInfo
MMC_NetworkScan
MMC_OpenUdpChannel
MMC_SetDefGateway
MMC_SetDhcp
MMC_SetIpAddr
MMC_SetIpMask
MMC_SetServerIp

11.1.1. MMC_CloseUdpChannel

Closes a UDP channel per RPC/IPC connection.

```
MMC_LIB_API int MMC_CloseUdpChannelCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_CLOSEUDPCHANNEL_IN* pInParam,
OUT MMC_CLOSEUDPCHANNEL_OUT* pOutParam
);
```

Motion Mode	NC – Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_CLOSEUDPCHANNEL_IN** input data structure using the MMC_CloseUdpChannel function.

pOutParam

Points to the **MMC_CLOSEUDPCHANNEL_OUT** output structure receiving information, as a result of calling the MMC_CloseUdpChannel function.

Remarks

None

Scope

Not limited

MMC_CLOSEUDPCHANNEL_IN Structure

typedef struct {
unsigned char dummy;
} MMC_CLOSEUDPCHANNEL_IN;

Parameters

dummy

Dummy IP address input. Any +ve character value.

MMC_CLOSEUDPCHANNEL_OUT Structure

typedef struct {
unsigned short usStatus;
short usErrorID;
}MMC_CLOSEUDPCHANNEL_OUT;

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-1 describes the function block for MMC_CloseUdpChannel

Figure 11-1: MMC_CloseUdpChannel function block

11.1.1.2. Function Block Code Example

```
int rc;
MMC_CLOSEUDPCHANNEL_IN     stCloseUDPChannel_in;
MMC_CLOSEUDPCHANNEL_OUT    stCloseUDPChannel_out;
//
// Inserting the structure parameters:
stCloseUDPChannel_in.dummy  = 1;             // dummy IP address
//
rc = MMC_CloseUdpChannelCmd (hConn, &stCloseUDPChannel_in, &stCloseUDPChannel_out);
if (rc != 0)
{
    HandleError();
}
```

11.1.2. MMC_GetDefGateway

Reads the default gateway IP address.

```
MMC_LIB_API int MMC_GetDefGatewayCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GET_DEFGATEWAY_IN* pInParam,
OUT MMC_GET_DEFGATEWAY_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GET_DEFGATEWAY_IN** input data structure using the MMC_GetDefGateway function.

pOutParam

Points to the **MMC_GET_DEFGATEWAY_OUT** output structure receiving information, as a result of calling the MMC_GetDefGateway function.

Remarks

None

Scope

Not limited

MMC_GET_DEFGATEWAY_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_GET_DEFGATEWAY_IN;
```

Parameters

dummy

Dummy IP address input. Any +ve character value 0 - 255.

MMC_GET_DEFGATEWAY_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned char cFirst;
unsigned char cSecond;
unsigned char cThird;
unsigned char cFourth;
}MMC_GET_DEFGATEWAY_OUT;
```

Parameters

cFirst

First octet gateway IP Address. Any three digit character number up to 255

cSecond

Second octet gateway IP Address. Any three digit character number up to 255

cThird

Third octet gateway IP Address. Any three digit character number up to 255

cFourth

Fourth octet gateway IP Address. Any three digit character number up to 255

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-2 describes the function block for MMC_GetDefGateway

Figure 11-2: MMC_GetDefGateway function block

11.1.2.2. Function Block Code Example

```
int rc;
MMC_GET_DEFGATEWAY_IN       stGetDefGateway_in;
MMC_GET_DEFGATEWAY_OUT      stGetDefGateway_out;
//
// Inserting the structure parameters:
stGetDefGateway_in.dummy  = 1;                // dummy IP address
//
rc = MMC_GetDefGatewayCmd (hConn, &stGetDefGateway_in, &stGetDefGateway_out);
printf("Default Gateway Status[%ld][%ld][%ld][%ld] ErrId[%d]\n", (long
int)stGetDefGateway_out.cFirst, (long int)stGetDefGateway_out.cSecond, (long
int)stGetDefGateway_out.cThird, (long int)stGetDefGateway_out.cFourth,
(short)stGetDefGateway_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.1.3. MMC_GetDhcp

Reads the DHCP mode.

```
MMC_LIB_API int MMC_GetDhcpCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GET_DHCP_IN* pInParam,
OUT MMC_GET_DHCP_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GET_DHCP_IN** input data structure using the MMC_GetDhcp function.

pOutParam

Points to the **MMC_GET_DHCP_OUT** output structure receiving information, as a result of calling the MMC_GetDhcp function.

Remarks

None

Scope

Not limited

MMC_GET_DHCP_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_GET_DHCP_IN;
```

Parameters

dummy

Default Gateway IP address input. Any +ve character value.

MMC_GET_DHCP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned char ucMode;
}MMC_GET_DHCP_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

ucMode

Mode is either static or dynamic. It is recommended to select Static and manually select the DHCP. If dynamic, the DHCP is automatically provided from the pool. Boolean values of TRUE/FALSE accepted

Figure 11-3 describes the function block for MMC_GetDhcp

Figure 11-3: MMC_GetDhcp function block

11.1.3.2. Function Block Code Example

```
int rc;
MMC_GET_DHCP_IN        stGetDHCP_in;
MMC_GET_DHCP_OUT       stGetDHCP_out;
//
// Inserting the structure parameters:
stGetDHCP_in.dummy  = 1;            // dummy IP address
printf("DHCP Status[%ld] ErrId[%d]\n", (long int)stGetDHCP_out.ucMode,
(short)stGetDHCP_out.usErrorID);
//
rc = MMC_GetDhcpCmd (hConn, &stGetDHCP_in, &stGetDHCP_out);
if (rc != 0)
{
    HandleError();
}
```

11.1.3.3. Implementation Example

11.1.4. MMC_GetIpAddr

Reads the DHCP mode.

```
MMC_LIB_API int MMC_GetIpAddrCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GET_IP_ADDRESS_IN* pInParam,
OUT MMC_GET_IP_ADDRESS_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GET_IP_ADDRESS_IN** input data structure using the MMC_GetIpAddr function.

pOutParam

Points to the **MMC_GET_IP_ADDRESS_OUT** output structure receiving information, as a result of calling the MMC_GetIpAddr function.

Remarks

None

Scope

Not limited

MMC_GET_IP_ADDRESS_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_GET_IP_ADDRESS_IN;
```

Parameters

dummy

Dummy IP address input. Any +ve character value 0 - 255.

MMC_GET_IP_ADDRESS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
char cFirst;
char cSecond;
char cThird;
char cFourth;
}MMC_GET_IP_ADDRESS_OUT;
```

Parameters

cFirst

First octet IP Address. Any three digit character number up to 255

cSecond

Second octet IP Address. Any three digit character number up to 255

cThird

Third octet IP Address. Any three digit character number up to 255

cFourth

Fourth octet IP Address. Any three digit character number up to 255

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-4 describes the function block for MMC_GetIpAddr

Figure 11-4: MMC_GetIpAddr function block

11.1.4.2. Function Block Code Example

```
int rc;
MMC_GET_IP_ADDRESS_IN      stGetIPAddress_in;
MMC_GET_IP_ADDRESS_OUT     stGetIPAddress_out;
//
// Inserting the structure parameters:
stGetIPAddress_in.dummy  = 1;           // dummy IP address
//
rc = MMC_GetIpAddrCmd (hConn, &stGetIPAddress_in, &stGetIPAddress_out);
printf("IP Address Status[%ld][%ld][%ld][%ld] ErrId[%d]\n", (long
int)stGetIPAddress_out.cFirst, (long int)stGetIPAddress_out.cSecond, (long
int)stGetIPAddress_out.cThird, (long int)stGetIPAddress_out.cFourth,
(short)stGetIPAddress_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.1.5. MMC_GetIpMask

Reads the IP mask.

```
MMC_LIB_API int MMC_GetIpMaskCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GET_IP_MASK_IN* pInParam,
OUT MMC_GET_IP_MASK_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GET_IP_MASK_IN** input data structure using the MMC_GetIpMask function.

pOutParam

Points to the **MMC_GET_IP_MASK_OUT** output structure receiving information, as a result of calling the MMC_GetIpMask function.

Remarks

None

Scope

Not limited

MMC_GET_IP_MASK_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_GET_IP_MASK_IN;
```

Parameters

dummy

Dummy IP mask input. Any +ve character value 0 - 255.

MMC_GET_IP_MASK_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
char cFirst;
char cSecond;
char cThird;
char cFourth;
}MMC_GET_IP_MASK_OUT;
```

Parameters

cFirst

First octet IP mask. Any three digit character number up to 255

cSecond

Second octet IP mask. Any three digit character number up to 255

cThird

Third octet IP mask. Any three digit character number up to 255

cFourth

Fourth octet IP mask. Any three digit character number up to 255

usStatus

Bitwise returned command status with the following values:

Aborted, Done, or CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-5 describes the function block for MMC_GetIpMask

Figure 11-5: MMC_GetIpMask function block

11.1.5.2. Function Block Code Example

```
int rc;
MMC_GET_IP_MASK_IN          stGetIPMask_in;
MMC_GET_IP_MASK_OUT         stGetIPMask_out;
//
// Inserting the structure parameters:
stGetIPMask_in.dummy   = 1;     // dummy IP address
//
rc = MMC_GetIpMaskCmd (hConn, &stGetIPMask_in, &stGetIPMask_out);
printf("IP Mask Status[%ld][%ld][%ld][%ld] ErrId[%d]\n", (long int)stGetIPMask_out.cFirst,
(long int)stGetIPMask_out.cSecond, (long int)stGetIPMask_out.cThird, (long
int)stGetIPMask_out.cFourth, (short)stGetIPMask_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.1.6. MMC_GetServerIp

Obtain the Server IP address.

```
MMC_LIB_API int MMC_GetServerIpCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GET_SERVERIP_IN* pInParam,
OUT MMC_GET_SERVERIP_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GET_SERVERIP_IN** input data structure using the MMC_GetServerIp function.

pOutParam

Points to the **MMC_GET_SERVERIP_OUT** output structure receiving information, as a result of calling the MMC_GetServerIp function.

Remarks

None

Scope

Not limited

MMC_GET_SERVERIP_IN Structure

```
typedef struct {
unsigned char dummy;
}MMC_GET_SERVERIP_IN;
```

Parameters

dummy

Server IP address exists. Boolean TRUE/FALSE values accepted.

MMC_GET_SERVERIP_OUT Structure

```
typedef struct{
int iPort;
unsigned short usStatus;
short usErrorID;
char cFirst;
char cSecond;
char cThird;
char cFourth;
}MMC_GET_SERVERIP_OUT;
```

Parameters

iPort

Port Address for the Server IP. Any +v integer.

cFirst

First octet IP address. Any three digit character number up to 255

cSecond

Second octet IP address. Any three digit character number up to 255

cThird

Third octet IP address. Any three digit character number up to 255

cFourth

Fourth octet IP address. Any three digit character number up to 255

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs**

on page 6-417 - 6-453.

Figure 11-6 describes the function block for MMC_GetServerIp

Figure 11-6: MMC_GetServerIp function block

11.1.6.2. Function Block Code Example

```
int rc;
MMC_GET_SERVERIP_IN        stGetServerIP_in;
MMC_GET_SERVERIP_OUT       stGetServerIP_out;
//
// Inserting the structure parameters:
stGetServerIP_in.dummy     = 1;     // dummy IP address
//
rc = MMC_GetServerIpCmd (hConn, &stGetServerIP_in, &stGetServerIP_out);
printf("Server Status[%ld][%ld][%ld][%ld] ErrId[%d]\n", (long int)stGetServerIP_out.cFirst,
(long int)stGetServerIP_out.cSecond, (long int)stGetServerIP_out.cThird, (long
int)stGetServerIP_out.cFourth, (short)stGetServerIP_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.1.7. MMC_NetworkInfo

Returns the network information, detailing the systems connected and/or defined in the resources file located in the G-MAS FLASH.

```
MMC_LIB_API int MMC_NetworkInfoCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_NETWORKINFO_IN* pInParam,
OUT MMC_NETWORKINFO_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_NETWORKINFO_IN** input data structure using the MMC_NetworkInfo function.

pOutParam

Points to the **MMC_NETWORKINFO_OUT** output structure receiving information, as a result of calling the MMC_NetworkInfo function.

Remarks

The network information may range from servo drives, Buses, etc. connected to the G-MAS via CANbus connections.

Scope

Not limited

MMC_NETWORKINFO_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_NETWORKINFO_IN;
```

Parameters

dummy

Dummy values

MMC_NETWORKINFO_OUT Structure

```
typedef struct{
int iBusType;
int iBusBaud;
int iNumOfActiveNodes;
int iNumOfResFileNodes;
unsigned short usStatus;
short usErrorID;
MMC_NODESINFO stNodesInfo[CAN_ID_MAX - 1];
}MMC_NETWORKINFO_OUT;
```

Parameters

iBusType

BUS type based on the following enumerator types:

eCOMM_TYPE_NONE =0

eCOMM_TYPE_ETHERCAT =1

eCOMM_TYPE_CAN =2

iBusType can have values of 0, 1, or 2

iBusBaud

BUS baud value according to the following list:

10, 20, 50, 100, 125, 250, 500, 800, 1000

Values of 1 – 1000 based on the specific values detailed in the description.

iNumOfActiveNodes

Number of active nodes in the resource file in the G-MAS and physically connected to the BUS.

iNumOfResFileNodes

Number of nodes in the resource files. Values of 1 – 127 accepted.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

stNodesInfo

Maximum no of all CANbus connected systems, excluding the G-MAS. The enumerator *stNodesInfo* is the Integer ID.

The array [CAN_ID_MAX - 1] can have values of [0 – 127].

The MMC_NODESINFO parameter defines the network nodes and their states. It has the following sub-parameter structure:

uiNodeType

Note: **This field is only valid for Elmo DS402 devices.**

32bit bitwise network node type with specific +ve integer values according to the following definition:

Bits	Description	
	Value	**Product**
0…7	0	Saxophone
	1	Clarinet
	2	Reserved
	3	Harmonica
	4	Cello
	5	Bassoon
	6	Trumpet
	7	Tuba
	8	Banjo
	9	Cornet
	10	Whistle
	11	Didge
	12	Tweeter
	13	Drum
	14	Reserved

	15	Bee
	16	Hornet
	17	Hawk
	18	Falcon
	19	Eagle
	20	Harp
	21	Guitar
	22	Reserved (for Bell)
	23	Trombone
	24	Panther
	25	Duo
	26 - 255	Reserved
8…11	Reserved for product name	
12…13	Always 0	
14	0 = SimplIQ 1 = Gold	
15	Always 0	
16	0: No CAN 1: CAN included	
17	0: EtherCAT active 1: TCP\IP	
18 - 20	Feedback type (Port A + Port B) more details see GoldGuitar brochure : 0: E type (Encoder + Encoder, Analog sensor) 1: A type (Absolute + Encoder, Analog sensor) 2: R type (Encoder, Analog sensor + Resolver) 3: M type (Absolute + Resolver)	
21 - 23	Reserved for HW type ("option in GoldGuitar" brochure)	
23 - 31	Reserved for general needs	

ucNodeID

Network Node ID. Numeric character values of 0 – 255 accepted.

ucCommType

The *ucCommType* enumerator communication node type. Can have the following values:

NODE_ELMO = 1

NODE_DS301 = 2

NODE_DS401 = 3

NODE_DS402= 4

NODE_DS406= 5

ucNodeState

The *ucNodeState* enumerator NodeState can have the following values:

eMMC_NODE_STATE_ENABLED = 0
Node exists on the CAN bus AND Node exists in the resource file.

eMMC_NODE_STATE_DISABLED
Node does not exist on the CAN bus AND Node exists in the resource file.

eMMC_NODE_STATE_UNKNOWN
Node exists on the CAN bus AND Node does not exist in the resource file.

dummy

The enumerator value of the node state. Character values of 0 – 255 accepted.

Figure 11-7 describes the function block for MMC_NetworkInfo

Figure 11-7: MMC_NetworkInfo function block

11.1.7.2. Function Block Code Example

```
int rc;
MMC_NETWORKINFO_IN          stNetworkInfo_in;
MMC_NETWORKINFO_OUT         stNetworkInfo_out;
//
// Inserting the structure parameters:
stNetworkInfo_in.dummy  = 1;    // dummy IP address
//
rc = MMC_NetworkInfoCmd (hConn, &stNetworkInfo_in, &stNetworkInfo_out);
if (rc != 0)
{
    HandleError();
}
```

11.1.8. MMC_NetworkScan

Scans the network to locate nodes on the network.

```
MMC_LIB_API int MMC_NetworkScanCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_NETWORKSCAN_IN* pInParam,
OUT MMC_NETWORKSCAN_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_NETWORKSCAN_IN** input data structure using the MMC_NetworkScan function.

pOutParam

Points to the **MMC_NETWORKSCAN_OUT** output structure receiving information, as a result of calling the MMC_NetworkScan function.

Remarks

None

Scope

Operation allowed in StandStill or Disabled state, but forbidden in any motion state.

Under certain circumstances, using this function in StandStill may cause the axis to enter Error Stop state.

MMC_NETWORKSCAN_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_NETWORKSCAN_IN;
```

Parameters

dummy

Server IP address exists. Boolean TRUE/FALSE values accepted.

MMC_NETWORKSCAN_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_NETWORKSCAN_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-8 describes the function block for MMC_NetworkScan

Figure 11-8: MMC_NetworkScan function block

11.1.8.2. Function Block Code Example

```
int rc;
MMC_NETWORKSCAN_IN          stNetworkScan_in;
MMC_NETWORKSCAN_OUT         stNetworkScan_out;
//
// Inserting the structure parameters:
//
rc = MMC_NetworkScanCmd (hConn, &stNetworkScan_in, &stNetworkScan_out);
if (rc != 0)
{
    HandleError();
}
```

11.1.9. MMC_OpenUdpChannel

Opens a UDP channel per RPC/IPC connection.

```
MMC_LIB_API int MMC_OpenUdpChannelCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_OPENUDPCHANNEL_IN* pInParam,
OUT MMC_OPENUDPCHANNEL_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_OPENUDPCHANNEL_IN** input data structure using the MMC_OpenUdpChannel function.

pOutParam

Points to the **MMC_OPENUDPCHANNEL_OUT** output structure receiving information, as a result of calling the MMC_OpenUdpChannel function.

Remarks

The UDP port has already been assigned in the MMC_InitConnection Function block.

Scope

Not limited

MMC_OPENUDPCHANNEL_IN Structure

```
typedef struct {
int iEventsMask;
int iPort;
char cFirst;
char cSecond;
char cThird;
char cFourth;
}MMC_OPENUDPCHANNEL_IN;
```

Parameters

iEventsMask

Describes the events mask for the callback function from the G-MAS to the host program. Defined according to the event IDs described in the section **12.18 Events Enumeration on page 12-1010**. +ve bitwise integer ID.

iPort

Port Address for the UDP Port Server IP. Any +ve integer.

cFirst

First octet IP address. Any three digit character number up to 255

cSecond

Second octet IP address. Any three digit character number up to 255

cThird

Third octet IP address. Any three digit character number up to 255

cFourth

Fourth octet IP address. Any three digit character number up to 255

MMC_OPENUDPCHANNEL_OUT Structure

```
typedef struct {
unsigned short usStatus;
short usErrorID;
}MMC_OPENUDPCHANNEL_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-9 describes the function block for MMC_OpenUdpChannel

Figure 11-9: MMC_OpenUdpChannel function block

11.1.9.2. Function Block Code Example

```
int rc;
MMC_OPENUDPCHANNEL_IN       stOpenUDPChannel_in;
MMC_OPENUDPCHANNEL_OUT      stOpenUDPChannel_out;
//
// Inserting the structure parameters:
stOpenUDPChannel_in.iEventsMask     = 101;      // Describes the events mask
stOpenUDPChannel_in.iPort           = 54;       // Selects the UDP Port
stOpenUDPChannel_in.cFirst          = 152;      // First octet IP address
stOpenUDPChannel_in.cSecond         = 93;       // Second octet IP address
stOpenUDPChannel_in.cThird          = 43;       // Third octet IP address
stOpenUDPChannel_in.cFourth         = 15;       // Fourth octet IP address
//
rc = MMC_OpenUdpChannelCmd (hConn, &stOpenUDPChannel_in, &stOpenUDPChannel_out);
if (rc != 0)
{
    HandleError();
}
```

11.1.10. MMC_SetDefGateway

Set default gateway IP address.

```
MMC_LIB_API int MMC_SetDefGatewayCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SET_DEFGATEWAY_IN* pInParam,
OUT MMC_SET_DEFGATEWAY_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SET_DEFGATEWAY_IN** input data structure using the MMC_SetDefGateway function.

pOutParam

Points to the **MMC_SET_DEFGATEWAY_OUT** output structure receiving information, as a result of calling the MMC_SetDefGateway function.

Remarks

When new, by default, the gateway IP address of the G-MAS is set to 192.168.1.1.

Scope

Not limited

MMC_SET_DEFGATEWAY_IN Structure

```
typedef struct{
char cFirst;
char cSecond;
char cThird;
char cFourth;
}MMC_SET_DEFGATEWAY_IN;
```

Parameters

cFirst

First octet gateway IP Address. Any three digit character number up to 255

cSecond

Second octet gateway IP Address. Any three digit character number up to 255

cThird

Third octet gateway IP Address. Any three digit character number up to 255

cFourth

Fourth octet gateway IP Address. Any three digit character number up to 255

MMC_SET_DEFGATEWAY_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SET_DEFGATEWAY_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-10 describes the function block for MMC_SetDefGateway

Figure 11-10: MMC_SetDefGateway function block

11.1.10.2. Function Block Code Example

```
int rc;
MMC_SET_DEFGATEWAY_IN       stSetDefGateway_in;
MMC_SET_DEFGATEWAY_OUT      stSetDefGateway_out;
//
// Inserting the structure parameters:
stSetDefGateway_in.cFirst       = 152;       // First octet IP address
stSetDefGateway_in.cSecond      = 93;        // Second octet IP address
stSetDefGateway_in.cThird       = 43;        // Third octet IP address
stSetDefGateway_in.cFourth      = 15;        // Fourth octet IP address
//
rc = MMC_SetDefGatewayCmd (hConn, &stSetDefGateway_in, &stSetDefGateway_out);
if (rc != 0)
{
    HandleError();
}
```

11.1.11. MMC_SetDhcp

Sets the DHCP Mode for the G-MAS.

```
MMC_LIB_API int MMC_SetDhcpCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SET_DHCP_IN* pInParam,
OUT MMC_SET_DHCP_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SET_DHCP_IN** input data structure using the MMC_SetDhcp function.

pOutParam

Points to the **MMC_SET_DHCP_OUT** output structure receiving information, as a result of calling the MMC_SetDhcp function.

Remarks

None

Scope

Not limited

MMC_SET_DHCP_IN Structure

```
typedef struct{
unsigned char ucMode;
}MMC_SET_DHCP_IN;
```

Parameters

ucMode

Mode is either static or dynamic. It is recommended to select Static and manually select the DHCP. If dynamic, the DHCP is automatically provided from the pool.

Values accepted are Boolean, TRUE/FALSE.

MMC_SET_DHCP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_NETWORKSCAN_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-11 describes the function block for MMC_SetDhcp

Figure 11-11: MMC_SetDhcp function block

11.1.11.2. Function Block Code Example

```
int rc;
MMC_SET_DHCP_IN        stSetDHCP_in;
MMC_SET_DHCP_OUT       stSetDHCP_out;
//
// Inserting the structure parameters:
stSetDHCP_in.ucMode  = 1;    // Mode is either static or dynamic
//
rc = MMC_SetDhcpCmd (hConn, &stSetDHCP_in, &stSetDHCP_out);
if (rc != 0)
{
    HandleError();
}
```

11.1.12. MMC_SetIpAddr

Sets the G-MAS IP address.

```
MMC_LIB_API int MMC_SetIpAddrCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SET_IP_ADDRESS_IN* pInParam,
OUT MMC_SET_IP_ADDRESS_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SET_IP_ADDRESS_IN** input data structure using the MMC_SetIpAddr function.

pOutParam

Points to the **MMC_SET_IP_ADDRESS_OUT** output structure receiving information, as a result of calling the MMC_SetIpAddr function.

Remarks

When new, by default, the IP address of the G-MAS is set to 192.168.1.3.

Scope

Not limited

MMC_SET_IP_ADDRESS_IN Structure

```
typedef struct{
char cFirst;
char cSecond;
char cThird;
char cFourth;
}MMC_SET_IP_ADDRESS_IN;
```

Parameters

cFirst

First octet IP Address. Any three digit character number up to 255

cSecond

Second octet IP Address. Any three digit character number up to 255

cThird

Third octet IP Address. Any three digit character number up to 255

cFourth

Fourth octet IP Address. Any three digit character number up to 255

MMC_SET_IP_ADDRESS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SET_IP_ADDRESS_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-12 describes the function block for MMC_SetIpAddr.

Figure 11-12: MMC_SetIpAddr function block

11.1.12.2. Function Block Code Example

```
int rc;
MMC_SET_IP_ADDRESS_IN      stSetIPAddress_in;
MMC_SET_IP_ADDRESS_OUT     stSetIPAddress_out;
//
// Inserting the structure parameters:
stSetIPAddress_in.cFirst   = 152;      // First octet IP address
stSetIPAddress_in.cSecond  = 93;       // Second octet IP address
stSetIPAddress_in.cThird   = 43;       // Third octet IP address
stSetIPAddress_in.cFourth  = 15;       // Fourth octet IP address
//
rc = MMC_SetIpAddrCmd (hConn, &stSetIPAddress_in, &stSetIPAddress_out);
if (rc != 0)
{
    HandleError();
}
```

11.1.13. MMC_SetIpMask

Set the IP netmask of the G-MAS.

```
MMC_LIB_API int MMC_SetIpMaskCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SET_IP_MASK_IN* pInParam,
OUT MMC_SET_IP_MASK_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SET_IP_MASK_IN** input data structure using the MMC_SetIpMask function.

pOutParam

Points to the **MMC_SET_IP_MASK_OUT** output structure receiving information, as a result of calling the MMC_SetIpMask function.

Remarks

When new, by default, the IP netmask address of the G-MAS is set to 255.255.255.0.

Scope

Not limited

MMC_SET_IP_MASK_IN Structure

```
typedef struct{
char cFirst;
char cSecond;
char cThird;
char cFourth;
}MMC_SET_IP_MASK_IN;
```

Parameters

cFirst

First octet IP address mask. Any three digit character number up to 255

cSecond

Second octet IP address mask. Any three digit character number up to 255

cThird

Third octet IP address mask. Any three digit character number up to 255

cFourth

Fourth octet IP address mask. Any three digit character number up to 255

MMC_SET_IP_MASK_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SET_IP_MASK_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-13 describes the function block for MMC_SetIpMask

Figure 11-13: MMC_SetIpMask function block

11.1.13.2. Function Block Code Example

```
int rc;
MMC_SET_IP_MASK_IN          stSetIPMask_in;
MMC_SET_IP_MASK_OUT         stSetIPMask_out;
//
// Inserting the structure parameters:
stSetIPMask_in.cFirst  = 252;        // First octet IP mask
stSetIPMask_in.cSecond = 183;        // Second octet IP mask
stSetIPMask_in.cThird  = 143;        // Third octet IP mask
stSetIPMask_in.cFourth = 115;        // Fourth octet IP mask
//
rc = MMC_SetIpMaskCmd (hConn, &stSetIPMask_in, &stSetIPMask_out);
if (rc != 0)
{
    HandleError();
}
```

11.1.14. MMC_SetServerIp

Set the Server IP address of the host.

```
MMC_LIB_API int MMC_SetServerIpCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SET_SERVERIP_IN* pInParam,
OUT MMC_SET_SERVERIP_OUT* pOutParam
);
```

Motion Mode	NC - Not Relevant	Distributed - Not Relevant
Source	GMAS\includes\MMC_network_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SET_SERVERIP_IN** input data structure using the MMC_SetServerIp function.

pOutParam

Points to the **MMC_SET_SERVERIP_OUT** output structure receiving information, as a result of calling the MMC_SetServerIp function.

Remarks

When new, by default, the host IP address is set to 192.168.1.2.

Scope

Not limited

MMC_SET_SERVERIP_IN Structure

```
typedef struct {
unsigned char cFirst;
unsigned char cSecond;
unsigned char cThird;
unsigned char cFourth;
}MMC_SET_SERVERIP_IN;
```

Parameters

cFirst

First octet IP address. Any three digit character number up to 255

cSecond

Second octet IP address. Any three digit character number up to 255

cThird

Third octet IP address. Any three digit character number up to 255

cFourth

Fourth octet IP address. Any three digit character number up to 255

MMC_SET_SERVERIP_OUT Structure

```
typedef struct {
unsigned short usStatus
short usErrorID;
}MMC_SET_SERVERIP_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-14 describes the function block for MMC_SetServerIp

Figure 11-14: MMC_SetServerIp function block

11.1.14.2. Function Block Code Example

```
int rc;
MMC_SET_SERVERIP_IN        stSetServerIP_in;
MMC_SET_SERVERIP_OUT       stSetServerIP_out;
//
// Inserting the structure parameters:
stSetServerIP_in.cFirst    = 148;       // First octet IP mask
stSetServerIP_in.cSecond   = 148;       // Second octet IP mask
stSetServerIP_in.cThird    = 0;         // Third octet IP mask
stSetServerIP_in.cFourth   = 1;         // Fourth octet IP mask
//
rc = MMC_SetServerIpCmd (hConn, &stSetServerIP_in, &stSetServerIP_out);
if (rc != 0)
{
    HandleError();
}
```

11.2. Host Communication

Host communications consists of Modbus communications and will in the future consist of further communication devices and protocols.

11.3. Modbus Communication Function Blocks

The Modbus interface allows the client to communicate using TCP/IP protocol with a G-MAS compiled program, and manipulate the functions of axis motions via the windows client Modbus program. Modbus always uses default port 502, to poll data from G-MAS, and update shared data.

Registry values adjust the movement of the drive, whereas Coil values act as switches, and are therefore 0 or 1. The server API operates the Modbus using specific Registry or Coil tables, which are:

- Read/write to register values
- Read/write to coils Boolean values
- Read input coils
- Close Modbus

This is performed using a specific thread in *MultiAxisControl* that listens to a specific port, for changes in values in registers/coils, through client's Modbus application. However, the Modbus Read values for both the coils and registry tables cannot be input from external sources.

The server can read/write to specific registers/coils and use the values of the Modbus registers/coils, to start/stop axes, or move engines to specific destinations using input parameters that come from Modbus. These parameters can be altered through the application connected to the G-MAS server.

The Modbus application can connect to a specific GMAS IP using a specific table ID opened using the G-MAS's client application, start address, number of available values, read/write registers, read/write coils, and via the Modbus application, alter running axes, and movement of axes through values shared between the GMAS client's C application, and the Windows Modbus application.

The Windows Modbus application can alter the table ID it uses, read/write registers, read/write coils, read inputs, and change the refresh rates of Modbus shared data displayed in the application.

The following Modbus communication function blocks are described:

Modbus Communication
MMC_MbusIsRunning
MMC_MbusReadCoilsTable
MMC_MbusReadHoldingRegisterTable
MMC_MbusReadInputsTable
MMC_MbusStartServer
MMC_MbusStopServer
MMC_MbusWriteCoilsTable
MMC_MbusWriteHoldingRegisterTable

11.3.1. MMC_MbusIsRunning

Signals that the Modbus connection is operational.

```
MMC_LIB_API int MMC_MbusIsRunning(
IN MMC_CONNECT_HNDL hConn,
IN MMC_MODBUSISRUNNING_IN* pInParam,
OUT MMC_MODBUSISRUNNING_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_host_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_MODBUSISRUNNING_IN** input data structure using the MMC_MbusIsRunning function.

pOutParam

Points to the **MMC_MODBUSISRUNNING_OUT** output structure receiving information, as a result of calling the MMC_MbusIsRunning function.

Remarks

This function block checks whether the Modbus thread is running, and returns the *isrunning* parameter with value 1, if the Modbus is running.

Scope

Not limited

MMC_MODBUSISRUNNING_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_MODBUSISRUNNING_IN;
```

Parameters

dummy

Modbus is connected to the server ID, with the following values:

MODBUS_NOT_STARTED = 0

MODBUS_RUNNING=1

MMC_MODBUSISRUNNING_OUT Structure

```
typedef struct{
unsigned short isrunning;
unsigned short usStatus;
short usErrorID;
}MMC_MODBUSISRUNNING_OUT;
```

Parameters

isrunning

Returns 1 if Modbus is running (MODBUS_RUNNING), otherwise 0 (MODBUS_NOT_STARTED).

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-15 describes the function block for MMC_MbusIsRunning

Figure 11-15: MMC_MbusIsRunning function block

11.3.1.2. Function Block Code Example

```
int rc;
MMC_MODBUSISRUNNING_IN      stMbusIsRunning_in;
MMC_MODBUSISRUNNING_OUT     stMbusIsRunning_out;
//
// Inserting the structure parameters:
stMbusIsRunning_in.dummy  = 1;            // Modbus is running (Boolean)
//
rc = MMC_MbusIsRunning (hConn, &stMbusIsRunning_in, &stMbusIsRunning_out);
if (rc != 0)
{
    HandleError();
}
```

11.3.2. MMC_MbusReadCoilsTable

Reads part of Modbus coils table.

```
MMC_LIB_API int MMC_MbusReadCoilsTable(
IN MMC_CONNECT_HNDL hConn,
IN MMC_MODBUSREADCOILS_IN* pInParam,
OUT MMC_MODBUSREADCOILS_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_host_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_MODBUSREADCOILS_IN** input data structure using the MMC_MbusReadCoilsTable function.

pOutParam

Points to the **MMC_MODBUSREADCOILS_OUT** output structure receiving information, as a result of calling the MMC_MbusReadCoilsTable function.

Remarks

Reads the coils table inside the Modbus where every value >0, is similar to Boolean value 1 in the coil. The function block variables include start reference, and reference count number of parameters to read. The internal output parameter is *coilsArr* with Modbus values.

Scope

Not limited

MMC_MODBUSREADCOILS_IN Structure

```
typedef struct{
int startRef;
int refCnt;
}MMC_MODBUSREADCOILS_IN;
```

Parameters

startRef

Start Reference from the base coil table of linear parameters. Any +ve integer values accepted

refCnt

Reference count. Any +ve integer values

MMC_MODBUSREADCOILS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
char coilsArr[MODBUS_IPC_READ_VALUES];
}MMC_MODBUSREADCOILS_OUT;
```

Parameters

coilsArr

Value of the coils array, with 250 as the maximum number of items to read from Modbus coils table. Array of +ve string values.

[MODBUS_IPC_READ_VALUES] has a an array value of [0….250]

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453.** -ve or +ve integer values

Figure 11-16 describes the function block for MMC_MbusReadCoilsTable

Figure 11-16: MMC_MbusReadCoilsTable function block

11.3.2.2. Function Block Code Example

```
int rc;
MMC_MODBUSREADCOILS_IN      stMbusReadCoils_in;
MMC_MODBUSREADCOILS_OUT     stMbusReadCoils_out;
//
// Inserting the structure parameters:
stMbusReadCoils_in.startRef     = 0;  // Start Reference from the base coil table of linear
parameters
stMbusReadCoils_in.refCnt       = 249;// Reference count

//
rc = MMC_MbusReadCoilsTable (hConn, &stMbusReadCoils_in, &stMbusReadCoils_out);
printf("Mbus Coils Table Status[%ld][%ld] ErrId[%d]\n", (long
int)stMbusReadCoils_out.coilsArr[0], (long int)stMbusReadCoils_out.coilsArr[1],
(short)stMbusReadCoils_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.3.3. MMC_MbusReadHoldingRegisterTable

Reads part of Modbus holding register table.

```
MMC_LIB_API int MMC_MbusReadHoldingRegisterTable(
IN MMC_CONNECT_HNDL hConn,
IN MMC_MODBUSREADHOLDINGREGISTERSTABLE_IN *pInParam,
OUT MMC_MODBUSREADHOLDINGREGISTERSTABLE_OUT *pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_host_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_MODBUSREADHOLDINGREGISTERSTABLE_IN** input data structure using the MMC_MbusReadHoldingRegisterTable function.

pOutParam

Points to the **MMC_MODBUSREADHOLDINGREGISTERSTABLE_OUT** output structure receiving information, as a result of calling the MMC_MbusReadHoldingRegisterTable function.

Remarks

Reads the registers table inside the Modbus, with parameters including, start reference, and reference count number of parameters to read. The internal output parameter is *regArr* with Modbus values.

Scope

Not limited

MMC_MODBUSREADHOLDINGREGISTERSTABLE_IN Structure

```
typedef struct{
int startRef;
int refCnt;
}MMC_MODBUSREADHOLDINGREGISTERSTABLE_IN;
```

Parameters

startRef

Start Reference from the base holding register table of linear parameters. Any +ve integer values accepted

refCnt

Reference count. Any +ve integer values

MMC_MODBUSREADHOLDINGREGISTERSTABLE_OUT Structure

```
typedef struct{
short regArr[MODBUS_IPC_READ_VALUES];
unsigned short usStatus;
short usErrorID;
}MMC_MODBUSREADHOLDINGREGISTERSTABLE_OUT;
```

Parameters

regArr

Displays the array values of the registry tables, with 250 as the maximum number of items to read from Modbus registry table. Array of +ve string values.

[MODBUS_IPC_READ_VALUES] has a an array value of [0….250]

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453.** -ve or +ve integer values

Figure 11-17 describes the function block for MMC_MbusReadHoldingRegisterTable

Figure 11-17: MMC_MbusReadHoldingRegisterTable function block

11.3.3.2. Function Block Code Example

```
int rc;
MMC_MODBUSREADHOLDINGREGISTERSTABLE_IN       stMbusReadHoldingTable_in;
MMC_MODBUSREADHOLDINGREGISTERSTABLE_OUT      stMbusReadHoldingTable_out;
//
// Inserting the structure parameters:
stMbusReadHoldingTable_in.startRef  = 0;//Start Reference from the base coil table of linear
parameters
stMbusReadHoldingTable_in.refCnt    = 249;// Reference count
//
rc = MMC_MbusReadHoldingRegisterTable (hConn, &stMbusReadHoldingTable_in,
&stMbusReadHoldingTable_out);
printf("Mbus Read Holding Register Table Status[%ld][%ld] ErrId[%d]\n", (long
int)stMbusReadHoldingTable_out.regArr[0], (long int)stMbusReadHoldingTable_out.regArr[1],
(short)stMbusReadHoldingTable_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.3.4. MMC_MbusReadInputsTable

Reads inputs to the Modbus Inputs Table.

```
MMC_LIB_API int MMC_MbusReadInputsTable(
IN MMC_CONNECT_HNDL hConn,
IN MMC_MODBUSREADINPUTS_IN *pInParam,
OUT MMC_MODBUSREADINPUTS_OUT *pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_host_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_MODBUSREADINPUTS_IN** input data structure using the MMC_MbusReadInputsTable function.

pOutParam

Points to the **MMC_MODBUSREADINPUTS_OUT** output structure receiving information, as a result of calling the MMC_MbusReadInputsTable function.

Remarks

Reads the Inputs table inside the Modbus, with parameters including, start reference, and reference count number of parameters to read. The internal output parameter is *InputsArr* with Modbus values. The inputs table cannot be changed by the Windows Modbus application, or using this API.

Scope

Not limited

MMC_MODBUSREADINPUTS_IN Structure

```
typedef struct{
int startRef;
int refCnt;
}MMC_MODBUSREADINPUTS_IN;
```

Parameters

startRef

Start reference from the base inputs table of linear parameters. Any +ve integer values accepted

refCnt

Reference count. Any +ve integer values

MMC_MODBUSREADINPUTS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
char inputsArr[MODBUS_IPC_READ_VALUES];
}MMC_MODBUSREADINPUTS_OUT;
```

Parameters

inputsArr

Value of the inputs array, with 250 as the maximum number of items to read from the Modbus inputs table. Array of +ve string values.

[MODBUS_IPC_READ_VALUES] has a an array value of [0….250]

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453.** -ve or +ve integer values

Figure 11-18 describes the function block for MMC_MbusReadInputsTable

Figure 11-18: MMC_MbusReadInputsTable function block

11.3.4.2. Function Block Code Example

```
int rc;
MMC_MODBUSREADINPUTS_IN         stMbusReadInputs_in;
MMC_MODBUSREADINPUTS_OUT        stMbusReadInputs_out;
//
// Inserting the structure parameters:
stMbusReadInputs_in.startRef    = 0;  // Start Reference from the base coil table of linear
parameters
stMbusReadInputs_in.refCnt      = 250;// Reference count
//
rc = MMC_MbusReadInputsTable (hConn, &stMbusReadInputs_in, &stMbusReadInputs_out);
printf("Mbus Read Inputs Table Status[%ld][%ld] ErrId[%d]\n", (long
int)stMbusReadInputs_out.inputsArr[0], (long int)stMbusReadInputs_out.inputsArr[1],
(short)stMbusReadInputs_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.3.5. MMC_MbusStartServer

Starts the Modbus server listening thread with an ID value as a parameter.

```
MMC_LIB_API int MMC_MbusStartServer(
IN MMC_CONNECT_HNDL hConn,
IN MMC_MODBUSSTARTSERVER_IN *pInParam,
OUT MMC_MODBUSSTARTSERVER_OUT *pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_host_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_MODBUSSTARTSERVER_IN** input data structure using the MMC_MbusStartServer function.

pOutParam

Points to the **MMC_MODBUSSTARTSERVER_OUT** output structure receiving information, as a result of calling the MMC_MbusStartServer function.

Remarks

None

Scope

Not limited

MMC_MODBUSSTARTSERVER_IN Structure

```
typedef struct{
unsigned short id;
}MMC_MODBUSSTARTSERVER_IN;
```

Parameters

id

Modbus start server enumerator ID has the following values:

MODBUS_NOT_STARTED = 0

MODBUS_RUNNING = 1

MODBUS_STOPPED = 2

After the G-MAS is powered-up, Modbus server is in the MODBUS_NOT_STARTED state – Initial state, the Modbus server does not exist.

After the Modbus server state is changed to MODBUS _RUNNING – the Modbus server is created, transmissions from Modbus clients will be handled by the server.

When the Modbus server state is changed to MODBUS_STOPPED – the Modbus server connection is removed, no transmissions will be handled from different Modbus clients.

MMC_MODBUSSTARTSERVER_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_MODBUSSTARTSERVER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453.** -ve or +ve integer values

Figure 11-19 describes the function block for MMC_MbusStartServer

Figure 11-19: MMC_MbusStartServer function block

11.3.5.2. Function Block Code Example

```
int rc;
MMC_MODBUSSTARTSERVER_IN        stMbusStartServer_in;
MMC_MODBUSSTARTSERVER_OUT       stMbusStartServer_out;
//
// Inserting the structure parameters:
stMbusStartServer_in.id    = 1;     // Modbus start server enumerator ID
//
rc = MMC_MbusStartServer (hConn, &stMbusStartServer_in, &stMbusStartServer_out);
if (rc != 0)
{
    HandleError();
}
```

11.3.6. MMC_MbusStopServer

Stops the Modbus server listening thread.

```
MMC_LIB_API int MMC_MbusStopServer(
IN MMC_CONNECT_HNDL hConn,
IN MMC_MODBUSSTOPSERVER_IN* pInParam,
OUT MMC_MODBUSSTOPSERVER_OUT *pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_host_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_MODBUSSTOPSERVER_IN** input data structure using the MMC_MbusStopServer function.

pOutParam

Points to the **MMC_MODBUSSTOPSERVER_OUT** output structure receiving information, as a result of calling the MMC_MbusStopServer function.

Remarks

None

Scope

Not limited

MMC_MODBUSSTOPSERVER_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_MODBUSSTOPSERVER_IN;
```

Parameters

dummy

Dummy Modbus stops server input. 0, 1, 2 integer values accepted.

MMC_MODBUSSTOPSERVER_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_MODBUSSTOPSERVER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-20 describes the function block for MMC_MbusStopServer

Figure 11-20: MMC_MbusStopServer function block

11.3.6.2. Function Block Code Example

```
int rc;
MMC_MODBUSSTOPSERVER_IN         stMbusStopServer_in;
MMC_MODBUSSTOPSERVER_OUT        stMbusStopServer_out;
//
// Inserting the structure parameters:
stMbusStopServer_in.dummy  = 0;       // Modbus stops server input
//
rc = MMC_MbusStopServer (hConn, &stMbusStopServer_in, &stMbusStopServer_out);
if (rc != 0)
{
    HandleError();
}
```

11.3.7. MMC_MbusWriteCoilsTable

Writes to part of Modbus coils table inside the Modbus where every parameter >0, is similar to Boolean value 1.

```
MMC_LIB_API int MMC_MbusWriteCoilsTable(
IN MMC_CONNECT_HNDL hConn,
IN MMC_MODBUSWRITECOILS_IN *pInParam,
OUT MMC_MODBUSWRITECOILS_OUT *pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_host_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_MODBUSWRITECOILS_IN** input data structure using the MMC_MbusWriteCoilsTable function.

pOutParam

Points to the **MMC_MODBUSWRITECOILS_OUT** output structure receiving information, as a result of calling the MMC_MbusWriteCoilsTable function.

Remarks

The function parameters include start reference, reference count number of parameters, and table of parameters.

Scope

Not limited

MMC_MODBUSWRITECOILS_IN Structure

```
typedef struct{
int startRef;
int refCnt;
char coilsArr[MODBUS_IPC_WRITE_VALUES];
}MMC_MODBUSWRITECOILS_IN;
```

Parameters

startRef

Start Reference from the base coil table of linear parameters. Any +ve integer values accepted

refCnt

Reference count. Any +ve integer values

coilsArr

Value of the coils array, with 250 as the maximum number of items to read from Modbus coils table. Array of +ve string values.

[MODBUS_IPC_READ_VALUES] has a an array value of [0….250]

MMC_MODBUSWRITECOILS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_MODBUSWRITECOILS_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453.** -ve or +ve integer values

Figure 11-21 describes the function block for MMC_MbusWriteCoilsTable

Figure 11-21: MMC_MbusWriteCoilsTable function block

11.3.7.2. Function Block Code Example

```
int rc;
MMC_MODBUSWRITECOILS_IN         stMbusWriteCoils_in;
MMC_MODBUSWRITECOILS_OUT        stMbusWriteCoils_out;
//
// Inserting the structure parameters:
stMbusWriteCoils_in.startRef        = 0;   // Start Reference from the base coil table of
linear parameters
stMbusWriteCoils_in.refCnt          = 249; // Reference count
stMbusWriteCoils_in.coilsArr[10]    = 2;   // Reference count
//
rc = MMC_MbusWriteCoilsTable (hConn, &stMbusWriteCoils_in, &stMbusWriteCoils_out);
if (rc != 0)
{
    HandleError();
}
```

11.3.8. MMC_MbusWriteHoldingRegisterTable

Writes to part of the Modbus register table inside the Modbus.

```
MMC_LIB_API int MMC_MbusWriteHoldingRegisterTable(
IN MMC_CONNECT_HNDL hConn,
IN MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN* pInParam,
OUT MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_host_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN** input data structure using the MMC_MbusWriteHoldingRegisterTable function.

pOutParam

Points to the **MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_OUT** output structure receiving information, as a result of calling the MMC_MbusWriteHoldingRegisterTable function.

Remarks

The function parameters include start reference, reference count number of parameters to write, and the register table with values to write into the Modbus table.

Scope

Not limited

MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN Structure

```
typedef struct{
int startRef;
int refCnt;
short regArr[MODBUS_IPC_WRITE_VALUES];
}MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN;
```

Parameters

startRef

Start Reference from the base coil table of linear parameters. Any +ve integer values accepted

refCnt

Reference count. Any +ve integer values

regArr

Array value of the register table, with 250 as the maximum number of items to write from the Modbus registry table.

[MODBUS_IPC_READ_VALUES] has a an array value of [0….250]

MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453.** -ve or +ve integer values

Figure 11-22 describes the function block for MMC_MbusWriteHoldingRegisterTable

Figure 11-22: MMC_MbusWriteHoldingRegisterTable function block

11.3.8.2. Function Block Code Example

```
int rc;
MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN      stMbusWriteHoldingRegTable_in;
MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_OUT     stMbusWriteHoldingRegTable_out;
//
// Inserting the structure parameters:
stMbusWriteHoldingRegTable_in.startRef   = 0;     // Start Reference from the base coil table
of linear parameters
stMbusWriteHoldingRegTable_in.refCnt     = 249;   // Reference count
stMbusWriteHoldingRegTable_in.regArr[10] = 65534; // Array value of the register table
//
rc = MMC_MbusWriteHoldingRegisterTable (hConn, &stMbusWriteHoldingRegTable_in,
&stMbusWriteHoldingRegTable_out);
if (rc != 0)
{
    HandleError();
}
```

11.4. CANbus Drive Communication

This section refers to the CANbus motion (DS-402) communication to the G-MAS. This form of communication uses the CANopen protocol and device profile specification for embedded systems used in automation. CANopen implements the layers above and includes the network layer. The CANopen standard consists of an addressing scheme, several small communication protocols, and an application layer defined by the device profile. The communication protocols support network management, device monitoring, and communication between nodes, including a simple transport layer for message segmentation/desegmentation. Elmo implements CANbus communications via the Process Data Object (PDO) and/or Service Data Object (SDO) protocols.

The physical layer of CANopen, CANbus, can only transmit short packages consisting of an 11-bit ID, remote transmission request (RTR) bit and 0 to 8 bytes of data. The CANopen standard divides the 11-bit CAN frame ID into a 4-bit function code and 7-bit CANopen node ID. This limits the number of devices in a CANopen network to 127. An extension to the CANbus standard (CAN 2.0 B) allows extended frame IDs of 29 bits. The 11-bit ID of a CAN-frame is called the Communication Object Identifier, or COB-ID. In case of a transmission collision, the bus arbitration used in the CANbus allows the frame with the smallest ID to be transmitted first and without a delay. Since in CANopen frames the first 4 bits of the frame ID are reserved for the function code, giving a low code number for time critical functions ensures the lowest possible delay.

The standard reserves certain COB-IDs to network management and SDO transfers. Some function codes and COB-IDs have to be mapped as standard functions after device initialization, but can be configured for other uses later.

The Process Data Object (PDO) protocol processes real time data among various nodes. Up to 8 bytes (64bits) of data per single PDO may be processed either from or to the G-MAS. Elmo uses PDOs of which PDO1 and PDO2 are reserved for internal use, allowing PDO3 and PDO4 for transmission and receiving of operational data.

There are two kinds of PDOs, transmit and receive PDOs (TPDO and RPDO, or as used in Elmo TX PDO and RX PDO). The former is for data coming from the servo drive and G-MAS and the latter is for data going to the G-MAS, and then servo drive. PDOs can be sent synchronously or asynchronously. Synchronous PDOs are sent after the SYNC message whereas asynchronous messages are sent after an internal or external trigger.

The SDO protocol is used to access set and read values from the object directory of the Host system (SDO server), PLC, or HMI Panel. The G-MAS (SDO client) always initiates communication with the SDO server. In CANopen terminology, communication is viewed from the SDO server, so that a read from an object directory results in an SDO upload and a write to directory is an SDO download.

As the object directory values can be larger than the 8 byte limit of a CAN frame, the SDO protocol implements segmentation and desegmentation of longer messages. There are two types of SDO protocols, SDO download/upload, and SDO Block download/upload. The SDO block transfer is a newer addition to the standard, which allows large amounts of data to be transferred with slightly less protocol overhead.

11.4.1. Master – Slave Relations

A CAN master (or client) is a controller that sends requests to nodes to respond to its commands. A CAN slave (or server) responds to the commands issued by the CAN master. The CAN protocol permits both single-master and multiple-master networks. Every servo drive has a unique ID in the range [1…127], but the network master does not require an ID. The slave servo drive never sends an unrequested message, other than emergencies, and only responds to messages addressed to its ID or to broadcast messages, which have an ID of 0. All messages sent by a servo drive are marked with its own ID.

11.4.2. CANopen DS-402 Modes of Operation

The G-MAS uses the CANopen DS-402 standard to operate the servo drives in the various defined user motion modes. The operation of the Drive (PDS - Power Drive System) depends on the activated mode of operation. The PDS may implement several modes of operation. Since it is not possible to operate the modes in parallel, the G-MAS system activates the required function by selecting a mode of operation, using the correct sequence of commands (compliant to the DS-402 standard). The G-MAS control device writes to the modes of operation object to select the operation mode. The drive device provides the modes of operation display object, to indicate the actual activated operation mode. These modes use a controlword, statusword, and/or setpoint, target position etc., which are mode-specific. They are specific background mapping PDOs, which are transparent to the user and operate in the background. The following ID numbered event modes of operation are used by the parameter ***ucMotionMode***, supported by the Drive as per the DS-402 specification:

Modes available	Enumerator	ID
No operational DS-402 mode	OPM402_NO	-1
Profile position mode	OPM402_PROFILE_POSITION_MODE	1
Homing mode	OPM402_HOMING_MODE	6
Interpolated position mode	OPM402_INTERPOLATED_POSITION_MODE	7
Profile velocity mode (partial e.g. servo drives)	OPM402_PROFILE_VELOCITY_MODE	3
Torque profile mode	OPM402_TORQUE_PROFILE_MODE	4
Velocity mode (e.g. frequency converter)		
Cyclic sync position mode (EtherCAT)	OPM402_CYCLIC_SYNC_POSITION_MODE	8
Cyclic sync velocity mode	OPM402_CYCLIC_SYNC_VELOCITY_MODE	9
Cyclic sync torque mode	OPM402_CYCLIC_SYNC_TORQUE_MODE	10

With the exception of the Homing mode, the listed modes of operation use setpoints. In addition, manufacturer-specific modes of operation may also be implemented. These are not limited to setpoints. This may be the case in the new Elmo servo drives, to support the extended Drive Profiler and motion modes available.

The G-MAS Axis (Node) State Management mechanism is responsible to operate theDS-402 axis in the correct operation mode, in order to implement the *requestedPLCMotion* Motion mode. This is true for both NC as well as Distributed motions.

11.4.3. PDO Mapping

The PDO mapping determines which CANopen objects are to be mapped as RX or TX PDOs and how each PDO is triggered. Four RX and TX PDOs are designated in the system, and the GMAS default PDO3 and PDO4, are not mapped at power up. The PDO mapping can only be performed in the pre-operational, operation states, but the RX or TX PDO itself can be a synchronized PDO, or event driven PDO. The type of PDO is determined by its communication type object:

1 Synchronized PDO – triggered by SYNC command

255 Event driven PDO – triggered by value change

PDO	1		2		3		4	
	RX	TX	RX	TX	RX	TX	RX	TX
PP mode	CW(255)+Target Position	SW(255)+Actual Position	Binary Interpreter		User configurable		User configurable	
PV mode	CW(255)+Target Vel	SW(255) + Actual Vel						
PT mode	CW(255)+Target Torque	SW(1) + Actual Torque						
IP mode	CW(1)+Target Position	SW(1) + Actual Position						
HM mode	CW(255)	SW(255)						

Figure 11-23: DS-402 PDO mapping table

Figure 11-23 displays the DS-402 PDO mapping table where the values in parentheses are the default values, per motion mode. The PDO mapping can be changed dynamically, as a function of the specific DS-402 motion mode. However, some of the available PDO mappings are hard-coded per DS-402 motion mode. The correct mapping is sent to the nodes when the DS-402 motion mode is changed.

The user configurable TX PDO's used with the parameter MC_PDO_TYPE_TXPDO can be configured to the event groups defined by the enumerator NC_COMM_EVENT_GROUP and the variable ucEventGroup(or ucEventType) (enumerator value). These event groups are divided into two sections:

Event Groups	Application values for the variable ucEventGroup
Regular Event Groups	MMC_CONFIGREGULARPARAMEVENTPDO3/4_IN inputs
User Event Groups	MMC_CONFIGUSERPARAMEVENTPDO3/4_IN inputs

The User groups are defined by the parameters received from the G-MAS via the EAS application and are included in the table below:

Enumerator value	Explanation
Regular Event Groups	
NC_COMM_EVENT_NO_GROUP	
NC_COMM_EVENT_GROUP1	Position + Velocity (32 + 32 bits)
NC_COMM_EVENT_GROUP2	Position + Digital Inputs (32 + 32 bits)
NC_COMM_EVENT_GROUP3	Digital Inputs + Velocity (32 + 32 bits)
NC_COMM_EVENT_GROUP4	Digital Inputs + Current (32 + 16 bits)
NC_COMM_EVENT_GROUP5	Current + Position (16 + 32 bits)
NC_COMM_EVENT_GROUP6	Current + Velocity (16 + 32 bits)
NC_COMM_EVENT_GROUP11	Digital Inputs (32 bits)
User Event Groups	**(All 32 + 32 bits)**
NC_COMM_EVENT_GROUP7	iUser_1 + Position
NC_COMM_EVENT_GROUP8	iUser_1 + Digital Inputs
NC_COMM_EVENT_GROUP9	fUser_1 + Position
NC_COMM_EVENT_GROUP10	fUser_1 + Digital Inputs
NC_COMM_EVENT_GROUP12	iUser_1 + iUserAux_1
NC_COMM_EVENT_GROUP13	iUser_2 + iUserAux_2
NC_COMM_EVENT_GROUP14	fUser_1 + fUserAux_1
NC_COMM_EVENT_GROUP15	fUser_2 + fUserAux_2
NC_COMM_EVENT_GROUP16	General Purpose group parameter1 (value 0)
NC_COMM_EVENT_GROUP17	General Purpose group parameter2 (value 1)

Once the TX PDO's are received by the G-MAS, they are automatically copied to the relevant parameter within the G-MAS node. If a User variable is used, they will be copied to the 2 x 2 Network Parameters (Float + Integer) available in the node. These can be used from the user program.

In addition, the user may choose to implement a callback event, by registering the callback, for a specific PDO per node. In this situation, the user sets one of the above parameters (Inputs, Position, and Velocity), then use one of the following functions to return to the last updated object:

- ReadActualPosition
- ReadActualVelocity
- ReadDigitalInputs
- ReadActualTorque

Just as the TX PDO's are sent from the servo drive to the G-MAS and then host server, similarly, the host server can send RX PDO's to the G-MAS and servo drive. The user configurable RX PDO's uses the parameter MC_PDO_TYPE_RPDO, and can be configured to the following event groups defined by the enumerator NC_COMM_RXEVENT_GROUP and variable ***ucEventGroup (or ucEventType)*** (enumerator value):

Enumerator value	Explanation
User Event Groups	
NC_COMM_RXEVENT_NO_GROUP	
NC_COMM_RXEVENT_GROUP1	iUser_3 + iUserAux_3
NC_COMM_RXEVENT_GROUP2	iUser_4 + iUserAux_4
NC_COMM_RXEVENT_GROUP3	fUser_3 + fUserAux_3
NC_COMM_RXEVENT_GROUP4	fUser_4 + fUserAux_4
NC_COMM_RXEVENT_GROUP5	iUser_3 + fUserAux_3
NC_COMM_RXEVENT_GROUP6	General Purpose group parameter1
NC_COMM_RXEVENT_GROUP7	General Purpose group parameter2

11.4.4. Using Event Groups 16 and 17

This section explains how to use the Event Groups 16 and 17 with the functions MMC_CfgUserParamEvPDO4Cmd or MMC_CfgUserParamEvPDO3Cmd. It is important to note that MMC_CfgUserParamEvPDO4Cmd or MMC_CfgUserParamEvPDO3Cmd do not support DS-406 or DS-401 nodes. Mapping a general PDO is not possible for DS-406 devices, and with a DS-401 node, it is necessary to use the same method as in DS-402, but only with the functions MMC_ConfigGeneralTPDO3 and MMC_ConfigGeneralTPDO3. The ***ucEventType*** parameter field may hold the value of event group 16 or 17.

1. Map the desired TPDO objects by manually sending SDOs to the node. This procedure should be performed in compliance with the device user manual and according to the CANopen standards.

When using event groups 16 and 17, the GMAS will not map anything by itself, but will only store the data arriving after the user has manually mapped the PDO, in conditions that allow extraction by the user.

2. Before mapping the final TPDO objects by sending the last SDO (that triggers the node to send the TPDOs), use the functions MMC_CfgUserParamEvPDO4Cmd or MMC_CfgUserParamEvPDO3Cmd with event groups 16 or 17 and the proper communication parameter defined in the mapping process for SYNC, ASYNC or "on event". All other inputs have no importance.
3. Send the last SDO to Trigger the node to send the TPDOs
4. From now on, the GMAS saves the data which the node sends.
5. The data can be extracted using the function MMC_PDOGeneralReadCmd with the following ucPDONum parameter values:

ucPDONum Value	Action
0	Extracts the data of event group 16,
1	Extracts the data of event group 17.

6. In addition, when the TPDO arrives it is possible to receive an event with the data, refer to the use of the functions MMC_CfgEventModePDO4Cmd and MMC_CfgEventModePDO3Cmd.

11.4.5. Servo Drive Sub-Index

The variable ***ucSubIndex*** uses a single integer to represent the User Integer (UI) and User Float (UF) of the servo drive, dependent on the Regular and User Event Group selected for the variable ***ucEventGroup (or ucEventType)***. Since PDO3 and PDO4 functions are available, ucSubIndex may be set for the PDO3 function to any UI[1], UI[2]… etc., and another UI value for the PDO4 function. Alternatively, the PDO3 function may be set to a UI value, and the PDO4 function set to a UF value (UF[1], UF[2], ..etc.). The limitation is the AxisRef of the servo drive.

11.4.6. SYNC and Time Stamp

The SYNC message has two purposes:

- Synchronize the operation of synchronous PDOs. Only synchronous TPDOs can be used to transmit data from SimplIQ digital servo drives upon receiving a SYNC message.
- Synchronize the motion clock of the servo drive with a clock in the network master (G-MAS server).

Synchronization is performed in conjunction with the Time Stamp message. The servo drive motion clock counts microseconds (regardless of the sampling time of the drive). It is cyclic and has 32 bits, and therefore completes a full cycle in 4,295 seconds (approximately 72 minutes). When the motion clocks of all connected servo drives are synchronized to the motion clock of the master (G-MAS server), multiple servo drives can perform complex synchronized motion with exact timing set by the network master.

The drives are synchronized by the transmission of a SYNC message, whose arrival time is captured by the drive. Upon receipt of the SYNC, the drive keys in its internal timer. A Time Stamp is a 32-bit message that contains the master internal clock generated, when the client’s own SYNC is received. The Time Stamp causes a clock synchronization cycle to be executed. The drive uses the Time Stamp as the absolute timer and adjusts its internal time in relation to the time keyed in at the last SYNC1. To synchronize the master and drive clocks to full precision, the synchronization process is filtered in order to ensure that the timing jitter of the time stamp process does not adversely affect the smoothness of the servo drive motion. It takes about 200 SYNC-Time Stamp pairs to ensure that all clocks are fully synchronized. COB-ID 256 (0x100) is a constant dedicated ID used

for this purpose. The master can send Time Stamps at any time. A Time Stamp always refers to the previous SYNC message and must come no later than 5 seconds after the relevant SYNC message.

11.4.7. CAN Bulk Upload

The CAN bulk upload feature is intended to optimize the process of upload of any data from drive to host (recorder data, personality, parameters, etc.). Currently, the whole upload process of data is managed by the host. For example, to upload the recording data buffer, it is necessary to send an SDO message, receive an SDO response, and retrieve the data via UDP. This is a tedious and awkward process that increases the network and bus load.

Elmo proposes that the G-MAS will manage the upload process upon request from the host, i.e. the host will send the "Begin Upload" command, and the G-MAS will upload the recording buffer. Afterwards, the host will be able to send the "Get Uploaded Data" command, and the data buffer will be returned immediately. During the whole process, a status can be retrieved using the "Get Upload Status" command, which will return to the user, the following data:

- Amount of data received
- Upload state (init/in progress/error/etc.)
- Communication error (if any)
- Process error (if any)

11.4.8. CAN – PDO, SDO Configurator

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. The user can configure in advance, PDOs that are mapped, and SDOs to be sent when the node is in preoperational state or in operational state. The configuration is performed via the EAS application.

11.5. CANbus Function Blocks

The following CANbus drive communication function blocks are described:

Drive Communication
MMC_CancelVirtualEncoder
MMC_CancelParamEvPDO3
MMC_CancelParamEvPDO4
MMC_CfgRegParamEvPDO3
MMC_CfgRegParamEvPDO4
MMC_CfgUserParamEvPDO3
MMC_CfgUserParamEvPDO4
MMC_ChangeDefaultPDOConfiguration
MMC_ChngOpMode
MMC_ConfigEventModePDO3
MMC_ConfigEventModePDO4
MMC_ConfigVirtualEncoder

MMC_GetAxisByCanId
MMC_GetPDOInfo
MMC_GetSyncTime
MMC_PDOGeneralRead
MMC_PDOGeneralWrite
MMC_ReceiveCANRawData
MMC_SendCANRawData
MMC_SendCmd
MMC_SetHeartBeatConsumer
MMC_SetSyncTime
MMC_StartBulkUpload
MMC_GetBulkUploadStatus
MMC_GetBulkUploadData

11.5.1. MMC_CancelVirtualEncoder

This function cancels a defined servo-drive as the virtual CAN encoder.

```
MMC_LIB_API int MMC_CancelVirtualEncoderCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CANCELVIRTUALENCODER_IN* pInParam,
OUT MMC_CANCELVIRTUALENCODER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_CANCELVIRTUALENCODER_IN** input data structure using the MMC_CancelVirtualEncoder function.

pOutParam

Points to the **MMC_CANCELVIRTUALENCODER_OUT** output structure receiving information, as a result of calling the MMC_CancelVirtualEncoder function.

Remarks

The virtual encoder is based on the DS406 CAN encoder protocol on the network. It allows the user to broadcast the position of a servo drive to the other servo drives connected to the G-MAS. Any of the servo-drives may be defined as the encoder, as each of the following functions is axis related:

- MMC_SetPosition
- MMC_ConfigVirtualEncoder
- MMC_CancelVirtualEncoder

Scope

The servo drives must set as a group address. SDO's must be set to manually 'listen' on the group address for the position. The SYNC function should be used to set the G-MAS.

MMC_CANCELVIRTUALENCODER_IN Structure

```
typedef struct mmc_cancelvirtualencoder_in{
unsigned char ucDummy;
}MMC_CANCELVIRTUALENCODER_IN;
```

Parameters

ucDummy

Dummy value. Any -ve or +ve character.

MMC_CANCELVIRTUALENCODER_OUT Structure

```
typedef struct mmc_cancelvirtualencoder_out{
unsigned short usStatus;
short usErrorID;
}MMC_CANCELVIRTUALENCODER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-24 describes the function block for MMC_CancelVirtualEncoder

Figure 11-24: MMC_CancelVirtualEncoder function block

11.5.1.2. Function Block Code Example

11.5.1.3. Implementation Example

11.5.2. MMC_CancelParamEvPDO3

Cancels the TPDO3 and RXPDO3 event processing.

```
MMC_LIB_API int MMC_CancelParamEvPDO3Cmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CANCELPARAMEVENTPDO3_IN* pInParam,
OUT MMC_CANCELPARAMEVENTPDO3_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CANCELPARAMEVENTPDO3_IN** input data structure using the MMC_CancelParamEvPDO3 function.

pOutParam

Points to the **MMC_CANCELPARAMEVENTPDO3_OUT** output structure receiving information as a result of calling the MMC_CancelParamEvPDO3 function.

Remarks

None

Scope

All

MMC_CANCELPARAMEVENTPDO3_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_CANCELPARAMEVENTPDO3_IN;
```

Parameters

dummy

Dummy input. Any +ve integer values accepted.

MMC_CANCELPARAMEVENTPDO3_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CANCELPARAMEVENTPDO3_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-25 describes the function block for MMC_CancelParamEvPDO3

Figure 11-25: MMC_CancelParamEvPDO3 function block

11.5.2.2. Function Block Code Example

```
int rc;
MMC_CANCELPARAMEVENTPDO3_IN     stCancelParamEventPDO3_in;
MMC_CANCELPARAMEVENTPDO3_OUT    stCancelParamEventPDO3_out;
//
// Inserting the structure parameters:
stCancelParamEventPDO3_in.dummy     = 0; //Any dummy inputs
//
rc = MMC_CancelParamEvPDO3Cmd (hConn, iAxisRef, &stCancelParamEventPDO3_in,
&stCancelParamEventPDO3_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.3. MMC_CancelParamEvPDO4

Cancels the TPDO4 and RXPDO4 event processing.

```
MMC_LIB_API int MMC_CancelParamEvPDO3Cmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CANCELPARAMEVENTPDO4_IN* pInParam,
OUT MMC_CANCELPARAMEVENTPDO4_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CANCELPARAMEVENTPDO4_IN** input data structure using the MMC_CancelParamEvPDO4 function.

pOutParam

Points to the **MMC_CANCELPARAMEVENTPDO4_OUT** output structure receiving information as a result of calling the MMC_CancelParamEvPDO4 function.

Remarks

None

Scope

All

MMC_CANCELPARAMEVENTPDO4_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_CANCELPARAMEVENTPDO4_IN;
```

Parameters

dummy

Dummy input. Any +ve integer values accepted.

MMC_CANCELPARAMEVENTPDO4_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CANCELPARAMEVENTPDO4_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-26 describes the function block for MMC_CancelParamEvPDO4

Figure 11-26: MMC_CancelParamEvPDO4 function block

11.5.3.2. Function Block Code Example

```
int rc;
MMC_CANCELPARAMEVENTPDO4_IN     stCancelParamEventPDO4_in;
MMC_CANCELPARAMEVENTPDO4_OUT    stCancelParamEventPDO4_out;
//
// Inserting the structure parameters:
stCancelParamEventPDO4_in.dummy     = 0; //Any dummy inputs
//
rc = MMC_CancelParamEvPDO4Cmd (hConn, iAxisRef, &stCancelParamEventPDO4_in,
&stCancelParamEventPDO4_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.4. MMC_CfgRegParamEvPDO3

Configures regular parameter event PDO3 according group type.

```
MMC_LIB_API int MMC_CfgRegParamEvPDO3Cmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGREGULARPARAMEVENTPDO3_IN* pInParam,
OUT MMC_CONFIGREGULARPARAMEVENTPDO3_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGREGULARPARAMEVENTPDO3_IN** input data structure using the MMC_CfgRegParamEvPDO3 function.

pOutParam

Points to the **MMC_CONFIGREGULARPARAMEVENTPDO3_OUT** output structure receiving information as a result of calling the MMC_CfgRegParamEvPDO3 function.

Remarks

None

Scope

The PDO3 cannot be mapped when the axis is in *ErrorStop* and MMC_Reset must be performed.

Make sure when using this function block in Distributed mode, the Sync Time is set, using the function MMC_SetSyncTime.

MMC_CONFIGREGULARPARAMEVENTPDO3_IN Structure

```
typedef struct{
unsigned int uiPDOCommParamEvent;
unsigned short usEventTimer;
unsigned char ucEventGroup;
unsigned char ucPDOCommParam;
}MMC_CONFIGREGULARPARAMEVENTPDO3_IN;
```

Parameters

uiPDOCommParamEvent

This parameter inserts a PDO Events mechanism in the GMAS for servo drive operations like emit, motion complete, motion started etc. When the operation is performed, an event is sent back to the G-MAS, and thereon to any connected host.

Any +ve integer in bitwise form is accepted.

usEventTimer

The timer for the PDO3 event. Has the following conditions depending on the drive specifications. Acceptable values are any integer in millisecs:

0 for Synchronous data

>0 for Asynchronous data

For the following PDO3 event times:

MC_PDO_TIMER_NON = 0

MC_PDO_TIMER_1_MILISEC = 1

MC_PDO_TIMER_2_MILISEC = 2

MC_PDO_TIMER_3_MILISEC = 3

MC_PDO_TIMER_4_MILISEC = 4

MC_PDO_TIMER_5_MILISEC = 5

MC_PDO_TIMER_10_MILISEC = 10

MC_PDO_TIMER_20_MILISEC = 20

MC_PDO_TIMER_25_MILISEC = 25

MC_PDO_TIMER_50_MILISEC = 50

MC_PDO_TIMER_100_MILISEC = 100

MC_PDO_TIMER_150_MILISEC = 150

MC_PDO_TIMER_200_MILISEC = 200

MC_PDO_TIMER_250_MILISEC = 250

MC_PDO_TIMER_255_MILISEC = 255

ucEventGroup

Defines which group of events are to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738** the correct definition to be used. Any +ve character values are acceptable.

ucPDOCommParam

PDO communications parameter. Has the following +ve character values:

PDO_COM_PARAM_SYNC 0x01

PDO_COM_PARAM_ASYNC 0xFF

PDO_COM_PARAM_EVENT 0xFE

PDO events are only possible when the input argument *ucPDOCommParam*, is PDO_COM_PARAM_EVENT.

MMC_CONFIGREGULARPARAMEVENTPDO3_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGREGULARPARAMEVENTPDO3_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-27 describes the function block for MMC_CfgRegParamEvPDO3

Figure 11-27: MMC_CfgRegParamEvPDO3 function block

11.5.4.2. Function Block Code Example

```
int rc;
MMC_CONFIGREGULARPARAMEVENTPDO3_IN      stConfigRegParamEventPDO3_in;
MMC_CONFIGREGULARPARAMEVENTPDO3_OUT     stConfigRegParamEventPDO3_out;
//
// Inserting the structure parameters:
stConfigRegParamEventPDO3_in.uiPDOCommParamEvent  = 0xDD;               // inserts a PDO
Events mechanism in the GMAS for servo drive operations
stConfigRegParamEventPDO3_in.usEventTimer         = 1;                  //Event timer
stConfigRegParamEventPDO3_in.ucEventGroup         = NC_COMM_EVENT_GROUP1;    //Defines which
group of events are to be transferred from the G MAS
stConfigRegParamEventPDO3_in.ucPDOCommParam       = 0xFE;               //PDO communications
parameter
//
rc = MMC_CfgRegParamEvPDO3Cmd (hConn, iAxisRef, &stConfigRegParamEventPDO3_in,
&stConfigRegParamEventPDO3_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.5. MMC_CfgRegParamEvPDO4

Configures regular parameter event PDO4 according group type.

```
MMC_LIB_API int MMC_CfgRegParamEvPDO4Cmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGREGULARPARAMEVENTPDO4_IN* pInParam,
OUT MMC_CONFIGREGULARPARAMEVENTPDO4_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGREGULARPARAMEVENTPDO4_IN** input data structure using the MMC_CfgRegParamEvPDO4 function.

pOutParam

Points to the **MMC_CONFIGREGULARPARAMEVENTPDO4_OUT** output structure receiving information as a result of calling the MMC_CfgRegParamEvPDO4 function.

Remarks

None

Scope

The PDO4 cannot be mapped when the axis is in *ErrorStop* and MMC_Reset must be performed.

Make sure, when using this function block in Distributed mode, the Sync Time is set, using the function MMC_SetSyncTime.

MMC_CONFIGREGULARPARAMEVENTPDO4_IN Structure

```
typedef struct{
unsigned int uiPDOCommParamEvent;
unsigned short usEventTimer;
unsigned char ucEventGroup;
unsigned char ucPDOCommParam;
}MMC_CONFIGREGULARPARAMEVENTPDO4_IN;
```

Parameters

uiPDOCommParamEvent

This parameter inserts a PDO Events mechanism in the GMAS for servo drive operations like emit, motion complete, motion started etc. When the operation is performed, an event is sent back to the G-MAS, and thereon to any connected host.

Any +ve integer in bitwise form is accepted.

usEventTimer

The timer for the PDO4 event. Has the following conditions depending on the drive specifications. Acceptable values are any integer in millisecs:

0 for Synchronous data

>0 for Asynchronous data

For the following PDO4 event times:

MC_PDO_TIMER_NON = 0

MC_PDO_TIMER_1_MILISEC = 1

MC_PDO_TIMER_2_MILISEC = 2

MC_PDO_TIMER_3_MILISEC = 3

MC_PDO_TIMER_4_MILISEC = 4

MC_PDO_TIMER_5_MILISEC = 5

MC_PDO_TIMER_10_MILISEC = 10

MC_PDO_TIMER_20_MILISEC = 20

MC_PDO_TIMER_25_MILISEC = 25

MC_PDO_TIMER_50_MILISEC = 50

MC_PDO_TIMER_100_MILISEC = 100

MC_PDO_TIMER_150_MILISEC = 150

MC_PDO_TIMER_200_MILISEC = 200

MC_PDO_TIMER_250_MILISEC = 250

MC_PDO_TIMER_255_MILISEC = 255

ucEventGroup

Defines which group of events are to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738** the correct definition to be used. Any +ve character values are acceptable.

ucPDOCommParam

PDO communications parameter. Has the following +ve character values:

PDO_COM_PARAM_SYNC 0x01

PDO_COM_PARAM_ASYNC 0xFF

PDO_COM_PARAM_EVENT 0xFE

PDO events are only possible when the input argument *ucPDOCommParam*, is PDO_COM_PARAM_EVENT.

MMC_CONFIGREGULARPARAMEVENTPDO4_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGREGULARPARAMEVENTPDO4_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-28 describes the function block for MMC_CfgRegParamEvPDO4

Figure 11-28: MMC_CfgRegParamEvPDO4 function block

11.5.5.2. Function Block Code Example

```
int rc;
MMC_CONFIGREGULARPARAMEVENTPDO4_IN      stConfigRegParamEventPDO4_in;
MMC_CONFIGREGULARPARAMEVENTPDO4_OUT     stConfigRegParamEventPDO4_out;
//
// Inserting the structure parameters:
stConfigRegParamEventPDO4_in.uiPDOCommParamEvent  = 0xDE;               //inserts a PDO Events
mechanism in the GMAS for servo drive operations
stConfigRegParamEventPDO4_in.usEventTimer         = 1;                  //Event timer
stConfigRegParamEventPDO4_in.ucEventGroup         = NC_COMM_EVENT_GROUP1;//Defines which group
of events are to be transferred from the G MAS
stConfigRegParamEventPDO4_in.ucPDOCommParam       = 0xFE;                    //PDO
communications parameter
//
rc = MMC_CfgRegParamEvPDO4Cmd (hConn, iAxisRef, &stConfigRegParamEventPDO4_in,
&stConfigRegParamEventPDO4_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.6. MMC_CfgUserParamEvPDO3

Configures user parameter event PDO3 according to group type.

```
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
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGUSERPARAMEVENTPDO3_IN** input data structure using the MMC_CfgUserParamEvPDO3 function.

pOutParam

Points to the **MMC_CONFIGUSERPARAMEVENTPDO3_OUT** output structure receiving information as a result of calling the MMC_CfgUserParamEvPDO3 function.

Remarks

None

Scope

For both NC and Distributed modes. To use this function in distributed mode with MC_PDO_COMM_ON_SYNC , the function block MMC_SetSyncTime should be called first.

MMC_CONFIGUSERPARAMEVENTPDO3_IN Structure

```
typedef struct{
unsigned int uiPDOCommParamEvent;
unsigned short usEventTimer;
unsigned char ucEventGroup;
unsigned char ucSubIndex;
unsigned char ucPDOCommParam;
unsigned char ucPDOType;
}MMC_CONFIGUSERPARAMEVENTPDO3_IN;
```

Parameters

uiPDOCommParamEvent

This parameter inserts a PDO Events mechanism in the GMAS for servo drive operations like emit, motion complete, motion started etc. When the operation is performed, an event is sent back to the G-MAS, and thereon to any connected host.

Any +ve integer in bitwise form is accepted.

usEventTimer

The timer for the PDO3 event. Has the following conditions depending on the drive specifications. Acceptable values are any integer in millisecs:

0 for Synchronous data

>0 for Asynchronous data

For the following PDO3 event times:

MC_PDO_TIMER_NON = 0

MC_PDO_TIMER_1_MILISEC = 1

MC_PDO_TIMER_2_MILISEC = 2

MC_PDO_TIMER_3_MILISEC = 3

MC_PDO_TIMER_4_MILISEC = 4

MC_PDO_TIMER_5_MILISEC = 5

MC_PDO_TIMER_10_MILISEC = 10

MC_PDO_TIMER_20_MILISEC = 20

MC_PDO_TIMER_25_MILISEC = 25

MC_PDO_TIMER_50_MILISEC = 50

MC_PDO_TIMER_100_MILISEC = 100

MC_PDO_TIMER_150_MILISEC = 150

MC_PDO_TIMER_200_MILISEC = 200

MC_PDO_TIMER_250_MILISEC = 250

MC_PDO_TIMER_255_MILISEC = 255

ucEventGroup

Defines which group of events are to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738** the correct definition to be used. Any +ve character values are acceptable.

ucSubIndex

Defines which index value signifies User Integer and User Float values of the servo drive. Refer to the section **11.4.4 Using Event Groups 16 and 17**. Any +ve character integers are accepted as values

ucPDOCommParam

PDO communications parameter. Has the following +ve character values:

PDO_COM_PARAM_SYNC 0x01

PDO_COM_PARAM_ASYNC 0xFF

PDO_COM_PARAM_EVENT 0xFE

PDO events are only possible when the input argument *ucPDOCommParam*, is PDO_COM_PARAM_EVENT.

ucPDOType

The direction of the PDO, according to the MC_PDO_TYPE_ENUM enumerator:

MC_PDO_TYPE_RPDO

MC_PDO_TYPE_TXPDO

This will be dependent on the value of the parameters ucEventGroup, and ucSubIndex.

MMC_CONFIGUSERPARAMEVENTPDO3_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGUSERPARAMEVENTPDO3_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-29 describes the function block for MMC_CfgUserParamEvPDO3

Figure 11-29: MMC_CfgUserParamEvPDO3 function block

11.5.6.2. Function Block Code Example

```
int rc;
MMC_CONFIGUSERPARAMEVENTPDO3_IN          stConfigUserParamEventPDO3_in;
MMC_CONFIGUSERPARAMEVENTPDO3_OUT         stConfigUserParamEventPDO3_out;
//
// Inserting the structure parameters:
stConfigUserParamEventPDO3_in.uiPDOCommParamEvent = 0xDD;                //inserts a PDO Events
mechanism in the GMAS for servo drive operations
stConfigUserParamEventPDO3_in.usEventTimer        = 2;                   //Event timer
stConfigUserParamEventPDO3_in.ucEventGroup        = NC_COMM_EVENT_GROUP7;//Defines which group
of events are to be transferred from the G-MAS
stConfigUserParamEventPDO3_in.ucSubIndex          = 2,1;                 //Defines which index
value signifies the group of events to be transferred from the G-MAS
stConfigUserParamEventPDO3_in.ucPDOCommParam      = 0xFE;                //PDO communications
parameter
stConfigUserParamEventPDO3_in.ucPDOType           = MC_PDO_TYPE_TXPDO;   //The direction of the
PDO
//
rc = MMC_CfgUserParamEvPDO3Cmd (hConn, iAxisRef, &stConfigUserParamEventPDO3_in,
&stConfigUserParamEventPDO3_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.7. MMC_CfgUserParamEvPDO4

Configures user parameter event PDO4 according to group type.

```
MMC_LIB_API int MMC_CfgUserParamEvPDO4Cmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGUSERPARAMEVENTPDO4_IN* pInParam,
OUT MMC_CONFIGUSERPARAMEVENTPDO4_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGUSERPARAMEVENTPDO4_IN** input data structure using the MMC_CfgUserParamEvPDO4 function.

pOutParam

Points to the **MMC_CONFIGUSERPARAMEVENTPDO4_OUT** output structure receiving information as a result of calling the MMC_CfgUserParamEvPDO4 function.

Remarks

None

Scope

For both NC and Distributed modes. To use this function in distributed mode with MC_PDO_COMM_ON_SYNC , the function block MMC_SetSyncTime should be called first.

MMC_CONFIGUSERPARAMEVENTPDO4_IN Structure

```
typedef struct{
unsigned int uiPDOCommParamEvent;
unsigned short usEventTimer;
unsigned char ucEventGroup;
unsigned char ucSubIndex;
unsigned char ucPDOCommParam;
unsigned char ucPDOType;
}MMC_CONFIGUSERPARAMEVENTPDO4_IN;
```

Parameters

uiPDOCommParamEvent

This parameter inserts a PDO Events mechanism in the GMAS for servo drive operations like emit, motion complete, motion started etc. When the operation is performed, an event is sent back to the G-MAS, and thereon to any connected host.

Any +ve integer in bitwise form is accepted.

usEventTimer

The timer for the PDO4 event. Has the following conditions depending on the drive specifications. Acceptable values are any integer in millisecs:

0 for Synchronous data

>0 for Asynchronous data

For the following PDO4 event times:

MC_PDO_TIMER_NON = 0

MC_PDO_TIMER_1_MILISEC = 1

MC_PDO_TIMER_2_MILISEC = 2

MC_PDO_TIMER_3_MILISEC = 3

MC_PDO_TIMER_4_MILISEC = 4

MC_PDO_TIMER_5_MILISEC = 5

MC_PDO_TIMER_10_MILISEC = 10

MC_PDO_TIMER_20_MILISEC = 20

MC_PDO_TIMER_25_MILISEC = 25

MC_PDO_TIMER_50_MILISEC = 50

MC_PDO_TIMER_100_MILISEC = 100

MC_PDO_TIMER_150_MILISEC = 150

MC_PDO_TIMER_200_MILISEC = 200

MC_PDO_TIMER_250_MILISEC = 250

MC_PDO_TIMER_255_MILISEC = 255

ucEventGroup

Defines which group of events are to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738** the correct definition to be used. Any +ve character values are acceptable.

ucSubIndex

Defines which index value signifies User Integer and User Float values of the servo drive. Refer to the section **11.4.4 Using Event Groups** 16 and 17

This section explains how to use the Event Groups 16 and 17 with the functions MMC_CfgUserParamEvPDO4Cmd or MMC_CfgUserParamEvPDO3Cmd. It is important to note that MMC_CfgUserParamEvPDO4Cmd or MMC_CfgUserParamEvPDO3Cmd do not support DS-406 or DS-401 nodes. Mapping a general PDO is not possible for DS-406 devices, and with a DS-401 node, it is necessary to use the same method as in DS-402, but only with the functions MMC_ConfigGeneralTPDO3 and MMC_ConfigGeneralTPDO3. The ***ucEventType*** parameter field may hold the value of event group 16 or 17.

1. Map the desired TPDO objects by manually sending SDOs to the node. This procedure should be performed in compliance with the device user manual and according to the CANopen standards.

When using event groups 16 and 17, the GMAS will not map anything by itself, but will only store the data arriving after the user has manually mapped the PDO, in conditions that allow extraction by the user.

2. Before mapping the final TPDO objects by sending the last SDO (that triggers the node to send the TPDOs), use the functions MMC_CfgUserParamEvPDO4Cmd or MMC_CfgUserParamEvPDO3Cmd with event groups 16 or 17 and the proper communication parameter defined in the mapping process for SYNC, ASYNC or "on event". All other inputs have no importance.
3. Send the last SDO to Trigger the node to send the TPDOs
4. From now on, the GMAS saves the data which the node sends.
5. The data can be extracted using the function MMC_PDOGeneralReadCmd with the following ucPDONum parameter values:

ucPDONum Value	Action
0	Extracts the data of event group 16,
1	Extracts the data of event group 17.

6. In addition, when the TPDO arrives it is possible to receive an event with the data, refer to the use of the functions MMC_CfgEventModePDO4Cmd and MMC_CfgEventModePDO3Cmd.

Servo Drive Sub-Index **on page 11-740**. Any +ve character integers are accepted as values

ucPDOCommParam

PDO communications parameter. Has the following +ve character values:

PDO_COM_PARAM_SYNC 0x01

PDO_COM_PARAM_ASYNC 0xFF

PDO_COM_PARAM_EVENT 0xFE

PDO events are only possible when the input argument *ucPDOCommParam*, is PDO_COM_PARAM_EVENT.

ucPDOType

The direction of the PDO, according to the MC_PDO_TYPE_ENUM enumerator:

MC_PDO_TYPE_RPDO

MC_PDO_TYPE_TXPDO

This will be dependent on the value of the parameters ucEventGroup, and ucSubIndex.

MMC_CONFIGUSERPARAMEVENTPDO4_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGUSERPARAMEVENTPDO4_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Displays an error code as -ve or +ve integers.

Figure 11-30 describes the function block for MMC_CfgUserParamEvPDO4.

Figure 11-30: MMC_CfgUserParamEvPDO4 function block

11.5.7.2. Function Block Code Example

```
int rc;
MMC_CONFIGUSERPARAMEVENTPDO4_IN          stConfigUserParamEventPDO4_in;
MMC_CONFIGUSERPARAMEVENTPDO4_OUT         stConfigUserParamEventPDO4_out;
//
// Inserting the structure parameters:
stConfigUserParamEventPDO4_in.uiPDOCommParamEvent = 0xCC;               //inserts a PDO Events
mechanism in the GMAS for servo drive operations
stConfigUserParamEventPDO4_in.usEventTimer        = 1;                  //Event timer
stConfigUserParamEventPDO4_in.ucEventGroup        = NC_COMM_EVENT_GROUP7;    //Defines which
group of events are to be transferred from the G-MAS
stConfigUserParamEventPDO4_in.ucSubIndex          = 2,1;                //Defines which index
value signifies the group of events to be transferred from the G-MAS
stConfigUserParamEventPDO4_in.ucPDOCommParam      = 0xFE;               //PDO communications
parameter
stConfigUserParamEventPDO4_in.ucPDOType           = MC_PDO_TYPE_TXPDO;  //The direction of the
PDO
//
rc = MMC_CfgUserParamEvPDO4Cmd (hConn, iAxisRef, &stConfigUserParamEventPDO4_in,
&stConfigUserParamEventPDO4_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.8. MMC_ChangeDefaultPDOConfiguration

Changes the default PDO communication parameter.

```
MMC_LIB_API int MMC_ChangeDefaultPDOConfiguration(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGPDOCOMMPARAM_IN* pInParam,
OUT MMC_CONFIGPDOCOMMPARAM_OUT *pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGPDOCOMMPARAM_IN** input data structure using the MMC_ChangeDefaultPDOConfiguration function.

pOutParam

Points to the **MMC_CONFIGPDOCOMMPARAM_OUT** output structure receiving information as a result of calling the MMC_ChangeDefaultPDOConfiguration function.

Remarks

Allows the communication parameter to be changed from Sync to Async and visa versa.

Scope

All

MMC_CONFIGPDOCOMMPARAM_IN Structure

```
typedef struct mmc_configpdocommparam_in{
unsigned char ucPDONum;
unsigned char ucPDODir;
unsigned char ucPDOCommParam;
}MMC_CONFIGPDOCOMMPARAM_IN;
```

Parameters

ucPDONum

Changes a specific PDO's communication from sync to async and visa versa. Allowed values are 1 to 4, representing the PDO1, PDO2, PDO3, and PDO4.

ucPDODir

Changes the RX or TX specific PDO communication. Allowed values are:

RXPDO

TXPDO

ucPDOCommParam

PDO communications parameter. Has the following +ve character values:

PDO_COM_PARAM_SYNC	0x01 (1)
PDO_COM_PARAM_ASYNC	0xFF (255)
PDO_COM_PARAM_EVENT	0xFE

PDO events are only possible when the input argument *ucPDOCommParam*, is PDO_COM_PARAM_EVENT.

MMC_CONFIGPDOCOMMPARAM_OUT Structure

```
typedef struct mmc_configpdocommparam_out{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGPDOCOMMPARAM_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-31 describes the function block for MMC_ChangeDefaultPDOConfiguration.

Figure 11-31: MMC_ChangeDefaultPDOConfiguration function block

11.5.8.2. Function Block Code Example

11.5.9. MMC_ChngOpMode

Changes the motion mode between NC and Distributed. This is previous determined in the DS-402 mode.

```
MMC_LIB_API int MMC_ChngOpMode(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CHANGEMOTIONMODE_IN* pInParam,
OUT MMC_CHANGEMOTIONMODE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed – Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CHANGEMOTIONMODE_IN** input data structure using the MMC_ChngOpMode function.

pOutParam

Points to the **MMC_CHANGEMOTIONMODE_OUT** output structure receiving information as a result of calling the MMC_ChngOpMode function.

Remarks

None

Scope

The motion mode must be changed in distributed state e.g. from position mode to velocity mode.

MMC_CHANGEMOTIONMODE_IN Structure

```
typedef struct{
unsigned char ucMotionMode;
}MMC_CHANGEMOTIONMODE_IN;
```

Parameters

ucMotionMode

Enumerator for the motion mode. The DS-402 motion modes and IDs available are:

Profile position mode = 1

Profiled velocity mode (partial) = 3

Homing Mode = 6

Interpolated position mode = 7

Any of the above +ve values accepted

MMC_CHANGEMOTIONMODE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
} MMC_CHANGEMOTIONMODE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-32 describes the function block for MMC_ChngOpMode.

Figure 11-32: MMC_ChngOpMode function block

11.5.9.2. Function Block Code Example

```
int rc;
MMC_CHANGEMOTIONMODE_IN    stChangeOpMode_in;
MMC_CHANGEMOTIONMODE_OUT   stChangeOpMode_out;
//
// Inserting the structure parameters:
stChangeOpMode_in.ucMotionMode  = 1; //Enumerator for the motion mode
//
rc = MMC_ChngOpMode (hConn, iAxisRef, &stChangeOpMode_in, &stChangeOpMode_out) ;
if (rc != 0)
{
    HandleError() ;
}
```

11.5.9.3. Implementation Example

11.5.10. MMC_ConfigEventModePDO3

Configures event mode for the PDO3 according group type.

```
MMC_LIB_API int MMC_CfgEventModePDO3Cmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGEVENTMODEPDO3_IN* pInParam,
OUT MMC_CONFIGEVENTMODEPDO3_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed – Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGEVENTMODEPDO3_IN** input data structure using the MMC_ConfigEventModePDO3 function.

pOutParam

Points to the **MMC_CONFIGEVENTMODEPDO3_OUT** output structure receiving information as a result of calling the MMC_ConfigEventModePDO3 function.

Remarks

None

Scope

All

MMC_CONFIGEVENTMODEPDO3_IN Structure

```
typedef struct{
unsigned char ucPDOEventMode;
}MMC_CONFIGEVENTMODEPDO3_IN;
```

Parameters

ucPDOEventMode

PDO3 event mode. This enumerator has the following values:

MC_PDO_EVENT_NO_NOTIF = 0,

MC_PDO_EVENT_CYCLIC_NOTIF,

MC_PDO_EVENT_IMMIDIATE_NOTIF

of which the default event type is MC_PDO_EVENT_NO_NOTIF. When using MC_PDO_EVENT_IMMIDIATE_NOTIF mode, no endian swap is created on the data.

MMC_CONFIGEVENTMODEPDO3_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGEVENTMODEPDO3_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-33 describes the function block for MMC_ConfigEventModePDO3

Figure 11-33: MMC_ConfigEventModePDO3 function block

11.5.10.2. Function Block Code Example

```
int rc;
MMC_CONFIGEVENTMODEPDO3_IN      stCfgEventModePDO3_in;
MMC_CONFIGEVENTMODEPDO3_OUT     stCfgEventModePDO3_out;
//
// Inserting the structure parameters:
stCfgEventModePDO3_in.ucPDOEventMode     = MC_PDO_EVENT_NO_NOTIF;        //PDO3 event mode
//
rc = MMC_CfgEventModePDO3Cmd (hConn, iAxisRef, &stCfgEventModePDO3_in,
&stCfgEventModePDO3_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.10.3. Implementation Example

11.5.11. MMC_ConfigEventModePDO4

Configures event mode for the PDO4 according group type.

```
MMC_LIB_API int MMC_CfgEventModePDO4Cmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGEVENTMODEPDO4_IN* pInParam,
OUT MMC_CONFIGEVENTMODEPDO4_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed – Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGEVENTMODEPDO4_IN** input data structure using the MMC_ConfigEventModePDO4 function.

pOutParam

Points to the **MMC_CONFIGEVENTMODEPDO4_OUT** output structure receiving information as a result of calling the MMC_ConfigEventModePDO4 function.

Remarks

None

Scope

All

MMC_CONFIGEVENTMODEPDO4_IN Structure

```
typedef struct{
unsigned char ucPDOEventMode;
}MMC_CONFIGEVENTMODEPDO4_IN;
```

Parameters

ucPDOEventMode

PDO4 event mode. This enumerator has the following values:

MC_PDO_EVENT_NO_NOTIF = 0,

MC_PDO_EVENT_CYCLIC_NOTIF,

MC_PDO_EVENT_IMMIDIATE_NOTIF

of which the default event type is MC_PDO_EVENT_NO_NOTIF. When using MC_PDO_EVENT_IMMIDIATE_NOTIF mode, no endian swap is created on the data.

MMC_CONFIGEVENTMODEPDO4_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGEVENTMODEPDO4_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-34 describes the function block for MMC_ConfigEventModePDO4

Figure 11-34: MMC_ConfigEventModePDO4 function block

11.5.11.2. Function Block Code Example

```
int rc;
MMC_CONFIGEVENTMODEPDO4_IN      stCfgEventModePDO4_in;
MMC_CONFIGEVENTMODEPDO4_OUT     stCfgEventModePDO4_out;
//
// Inserting the structure parameters:
stCfgEventModePDO4_in.ucPDOEventMode     = MC_PDO_EVENT_NO_NOTIF;        //PDO4 event mode
//
rc = MMC_CfgEventModePDO4Cmd (hConn, iAxisRef, &stCfgEventModePDO4_in,
&stCfgEventModePDO4_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.12. MMC_ConfigVirtualEncoder

This function defines a servo-drive as the virtual CAN encoder.

```
MMC_LIB_API int MMC_ConfigVirtualEncoderCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGVIRTUALENCODER_IN* pInParam,
OUT MMC_CONFIGVIRTUALENCODER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGVIRTUALENCODER_IN** input data structure using the MMC_ConfigVirtualEncoder function.

pOutParam

Points to the **MMC_CONFIGVIRTUALENCODER_OUT** output structure receiving information, as a result of calling the MMC_ConfigVirtualEncoder function.

Remarks

The virtual encoder is based on the DS406 CAN encoder protocol on the network. It allows the user to broadcast the position of a servo drive to the other servo drives connected to the G-MAS. Any of the servo-drives may be defined as the encoder, as each of the following functions is axis related:

- MMC_SetPosition
- MMC_ConfigVirtualEncoder
- MMC_CancelVirtualEncoder

Scope

The servo drives must set as a group address. SDO's must be set to manually 'listen' on the group address for the position. The SYNC function should be used to set the G-MAS. It should be noted that the group ID is limited to between 1 to 127.

MMC_CONFIGVIRTUALENCODER_IN Structure

```
typedef struct MMC_CONFIGVIRTUALENCODER_IN{
double dbLowPos;
double dbHighPos;
float fFactor;
unsigned char ucMode;
unsigned char ucGroupID;
}MMC_CONFIGVIRTUALENCODER_IN;
```

Parameters

dbLowPos

Low range of the virtual encoder. Any -ve or +ve double values in technical unit [u]

dbHighPos

High range of the virtual encoder. Any -ve or +ve double values in technical unit [u]

fFactor

Encoder factor. Any +ve integer value accepted.

ucMode

Defines the virtual encoder mode of the encoder with the following options:

NC_NODE_VIRTUAL_ENC_MODE_DISABLED = 0

NC_NODE_VIRTUAL_ENC_MODE_TARGET_POS

NC_NODE_VIRTUAL_ENC_MODE_ACTUAL_POS

ucGroupID

Group CAN ID. +ve integer accepted. The group ID is limited to between 1 to 127.

MMC_CONFIGVIRTUALENCODER_OUT Structure

```
typedef struct MMC_CONFIGVIRTUALENCODER_OUT{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGVIRTUALENCODER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-35 describes the function block for MMC_ConfigVirtualEncoder

Figure 11-35: MMC_ConfigVirtualEncoder function block

11.5.12.2. Function Block Code Example

11.5.13. MMC_GetAxisByCanId

Obtains axis handle according to the CANbus identity.

```
MMC_LIB_API int MMC_GetAxisByCanIdCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETAXISREFFROMCANID_IN* pInParam,
OUT MMC_GETAXISREFFROMCANID_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed – Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETAXISREFFROMCANID_IN** input data structure using the MMC_GetAxisByCanId function.

pOutParam

Points to the **MMC_GETAXISREFFROMCANID_OUT** output structure receiving information as a result of calling the MMC_GetAxisByCanId function.

Remarks

None

Scope

All

MMC_GETAXISREFFROMCANID_IN Structure

```
typedef struct{
unsigned char ucNodeID;
}MMC_GETAXISREFFROMCANID_IN;
```

Parameters

ucNodeID

Node ID from the CANbus. Any +ve integer accepted.

MMC_GETAXISREFFROMCANID_OUT Structure

```
typedef struct{
unsigned short usAxisRef;
unsigned short usStatus;
short usErrorID;
}MMC_GETAXISREFFROMCANID_OUT;
```

Parameters

usAxisRef

Axis reference. Any +ve bitwise integer.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-36 describes the function block for MMC_GetAxisByCanId

Figure 11-36: MMC_GetAxisByCanId function block

11.5.13.2. Function Block Code Example

```
int rc;
MMC_GETAXISREFFROMCANID_IN      stGetAxisRefFromCANID_in;
MMC_GETAXISREFFROMCANID_OUT     stGetAxisRefFromCANID_out;
//
// Inserting the structure parameters:
stGetAxisRefFromCANID_in.ucNodeID   = 4; //Node ID from the CANbus
//
rc = MMC_GetAxisByCanIdCmd (hConn, &stGetAxisRefFromCANID_in, &stGetAxisRefFromCANID_out);
printf("Axis By CAN ID Status[%ld] ErrId[%d]\n", (long
int)stGetAxisRefFromCANID_out.usAxisRef, (short)stGetAxisRefFromCANID_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.5.14. MMC_GetPDOInfo

Obtains the PDO information

```
MMC_LIB_API int MMC_GetPDOInfoCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GETPDOINFO_IN* pInParam,
OUT MMC_GETPDOINFO_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed – Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETPDOINFO_IN** input data structure using the MMC_GetPDOInfo function.

pOutParam

Points to the**MMC_GETPDOINFO_OUT** output structure receiving information as a result of calling the MMC_GetPDOInfo function.

Remarks

None

Scope

All

MMC_GETPDOINFO_IN Structure

```
typedef struct{
unsigned char ucPDONumber;
}MMC_GETPDOINFO_IN;
```

Parameters

ucPDONumber

The PDO index number. Any +ve character acceptable.

MMC_GETPDOINFO_OUT Structure

```
typedef struct{
int iPDOEventMode;
unsigned short usStatus;
short usErrorID;
unsigned char ucPDOCommType;
unsigned char ucTPDOCommEventGroup;
unsigned char ucRPDOCommEventGroup;
}MMC_GETPDOINFO_OUT;
```

Parameters

iPDOEventMode

The PDO event mode integer

ucPDOCommType

PDO communication type as a +ve character value

ucTPDOCommEventGroup

TPDO communication event group as a +ve character value

ucRPDOCommEventGroup

RPDO communication event group as a +ve character value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-37 describes the function block for MMC_GetPDOInfo

Figure 11-37: MMC_GetPDOInfo function block

11.5.14.2. Function Block Code Example

11.5.15. MMC_GetSyncTime

Where CANbus communication is relevant, returns the SYNC time.

```
MMC_LIB_API int MMC_GetSyncTimeCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETSYNCTIME_IN* pInParam,
OUT MMC_GETSYNCTIME_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETSYNCTIME_IN** input data structure using the MMC_GetSyncTime function.

pOutParam

Points to the **MMC_GETSYNCTIME_OUT** output structure receiving information as a result of calling the MMC_GetSyncTime function.

Remarks

None

Scope

All

MMC_GETSYNCTIME_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_GETSYNCTIME_IN;
```

Parameters

dummy

Dummy Sync time input. Any +ve values accepted.

MMC_GETSYNCTIME_OUT Structure

```
typedef struct{
unsigned short usSYNCTime;
unsigned short usStatus;
short usErrorID;
}MMC_GETSYNCTIME_OUT;
```

Parameters

usSYNCTime

Sync time output. Refer to the explanation in section **11.4.6 SYNC and Time Stamp on page 11-741**. Any +ve integer value between 0 to 1000.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-38 describes the function block for MMC_GetSyncTime

Figure 11-38: MMC_GetSyncTime function block

11.5.15.2. Function Block Code Example

```
int rc;
MMC_GETSYNCTIME_IN          stGetSYNCTime_in;
MMC_GETSYNCTIME_OUT         stGetSYNCTime_out;
//
// Inserting the structure parameters:
stGetSYNCTime_in.dummy      = 1; //Any dummy inputs
//
rc = MMC_GetSyncTimeCmd (hConn, &stGetSYNCTime_in, &stGetSYNCTime_out);
printf("Sync Time Status[%ld] ErrId[%d]\n", (long int)stGetSYNCTime_out.usSYNCTime,
(short)stGetSYNCTime_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.5.16. MMC_PDOGeneralRead

Reads a specific PDO message command.

```
MMC_LIB_API int MMC_PDOGeneralReadCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GENERALPARAMPDOREAD_IN* pInParam,
OUT MMC_GENERALPARAMPDOREAD_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GENERALPARAMPDOREAD_IN** input data structure using the MMC_PDOGeneralRead function.

pOutParam

Points to the **MMC_GENERALPARAMPDOREAD_OUT** output structure receiving information as a result of calling the MMC_PDOGeneralRead function.

Remarks

Reads a specific package of data sent

Scope

To gain the correct feedback data, make sure to set the DS-401 PD03 and/or PDO4 at the drives and G-MAS as per requirements. Refer to the sections **11.12.1 MMC_CancelGeneralRPDO3** to **11.12.8 MMC_ConfigGeneralTPDO4 on page 11-957 - 11-978.**

MMC_GENERALPARAMPDOREAD_IN Structure

```
typedef struct{
unsigned char ucParam;
}MMC_GENERALPARAMPDOREAD_IN;
```

Parameters

ucParam

Defines which of the general group of 32bit assigned events 16 or 17 is to be transferred to the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. Use the values 0, 1 denoted below to represent the assignment:

NC_COMM_EVENT_GROUP16 0

NC_COMM_EVENT_GROUP17 1

MMC_GENERALPARAMPDOREAD_OUT Structure

```
typedef struct{
#ifdef WIN32
unsigned __int64 ulliVal;
#else
unsigned long long int ulliVal;
#endif
unsigned short usStatus;
short usErrorID;
}MMC_GENERALPARAMPDOREAD_OUT;
```

Parameters

__int64 ulliVal or ulliVal

If function is defined for WIN32 then use *__int64 ulliVal*, else use *ulliVal*. Any +ve, –ve (Win32) or +ve 64bit (8 bytes) character and/or integer.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-39 describes the function block for MMC_PDOGeneralRead

Figure 11-39: MMC_PDOGeneralRead function block

11.5.16.2. Function Block Code Example

11.5.17. MMC_PDOGeneralWrite

Writes a specific PDO message command.

```
MMC_LIB_API int MMC_PDOGeneralWriteCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GENERALPARAMPDOWRITE_IN* pInParam,
OUT MMC_GENERALPARAMPDOWRITE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GENERALPARAMPDOWRITE_IN** input data structure using the MMC_PDOGeneralWrite function.

pOutParam

Points to the **MMC_GENERALPARAMPDOWRITE_OUT** output structure receiving information as a result of calling the MMC_PDOGeneralWrite function.

Remarks

Writes a specific data package to be sent.

Scope

To gain the correct feedback data, make sure to set the DS-401 PD03 and/or PDO4 at the drives and G-MAS as per requirements. Refer to the sections **11.12.1 MMC_CancelGeneralRPDO3** to **11.12.8 MMC_ConfigGeneralTPDO4 on page 11-957 - 11-978.**

MMC_GENERALPARAMPDOWRITE_IN Structure

```
typedef struct{
#ifdef WIN32
unsigned __int64 ulliVal;
#else
unsigned long long int ulliVal;
#endif
unsigned char ucParam;
}MMC_GENERALPARAMPDOWRITE_IN;
```

Parameters

__int64 ulliVal or ulliVal

If function is defined for WIN32 then use *__int64 ulliVal*, else use *ulliVal*. Any +ve, –ve (Win32) or +ve 64bit (8 bytes) character and/or integer.

ucParam

Defines which of the general group of 32bit assigned events 16 or 17 is to be transferred to the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. Use the values 0, 1 denoted below to represent the assignment:

NC_COMM_EVENT_GROUP16 0

NC_COMM_EVENT_GROUP17 1

MMC_GENERALPARAMPDOWRITE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_GENERALPARAMPDOWRITE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-40 describes the function block for MMC_PDOGeneralWrite

Figure 11-40: MMC_PDOGeneralWrite function block

11.5.17.2. Function Block Code Example

11.5.18. MMC_ReceiveCANRawData

Receives prepared CANopen RAW data (DS-301 or DS-402).

```
MMC_LIB_API int MMC_ReceiveCANRawData(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN unsigned short iTimeOutms,
OUT MMC_CAN_REPLY_DATA_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **iTimeOutms** input data structure using the MMC_ReceiveCANRawData function.

pOutParam

Points to the **MMC_CAN_REPLY_DATA_OUT** output structure receiving information as a result of calling the MMC_ReceiveCANRawData function.

Remarks

None

Scope

All

iTimeOutms Input

[IN] Time out delay to receive data. Integer time value in ms.

MMC_CAN_REPLY_DATA_OUT Structure

```
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];
}MMC_CAN_REPLY_DATA_OUT;
```

Parameters

usFunctionID

Request function ID of the raw data. Any +ve bitwise integer

usNumerator

Response to the send data. Any +ve bitwise integer

usDatasize

Size of the data sent. Any +ve bitwise integer

usPadding

Alignment padding of data. This parameter is not in use.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

usCOB_ID

11-bit ID of the CAN-frame. Any +ve bitwise number

usAxisRef

Axis reference. Any +ve bitwise integer.

can_data_length

Length of the CAN data. Any character integer value.

data[8]

Data. Character value with a maximum length of 8 bits.

Figure 11-41 describes the function block for MMC_ReceiveCANRawData

Figure 11-41: MMC_ReceiveCANRawData function block

11.5.18.2. Function Block Code Example

```
int rc;
MMC_CAN_REPLY_DATA_OUT     stCANReplyData_out;
//
// Inserting the structure parameters:
iTimeOutms = 1001;     //Time delay
//
rc = MMC_ReceiveCANRawData (hConn, iAxisRef, iTimeOutms, &stCANReplyData_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.19. MMC_SendCANRawData

Sends prepared CANopen RAW data (DS-301 or DS-402).

```
MMC_LIB_API int MMC_SendCANRawData(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_SENDRAWDATA_IN* pInParam,
OUT MMC_SENDRAWDATA_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_SENDRAWDATA_IN** input data structure using the MMC_SendCANRawData function.

pOutParam

Points to the **MMC_SENDRAWDATA_OUT** output structure receiving information as a result of calling the MMC_SendCANRawData function.

Remarks

None

Scope

All

MMC_SENDRAWDATA_IN Structure

```
typedef struct{
unsigned short usCOB_ID;
unsigned char ucLength;
unsigned char pData[8];
}MMC_SENDRAWDATA_IN;
```

Parameters

usCOB_ID

11-bit ID of the CAN-frame. Any +ve bitwise number

ucLength

Length of the raw data. Any +ve character value

pData[8]

Array raw data input dependant on the array size with a maximum array length of 8 bytes. Character value with a maximum length of 8 bits.

MMC_SENDRAWDATA_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SENDRAWDATA_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-42 describes the function block for MMC_SendCANRawData

Figure 11-42: MMC_SendCANRawData function block

11.5.19.2. Function Block Code Example

```
int rc;
MMC_SENDRAWDATA_IN     stSendRawData_in;
MMC_SENDRAWDATA_OUT    stSendRawData_out;
//
// Inserting the structure parameters:
stSendRawData_in.usCOB_ID  = 111;        //11-bit ID of the CAN-frame
stSendRawData_in.ucLength  = 8;          //Length of the raw data
stSendRawData_in.pData[0]  = 101;        //8 byte raw data
stSendRawData_in.pData[1]  = 100;        //8 byte raw data
stSendRawData_in.pData[2]  = 110;        //8 byte raw data
//
rc = MMC_SendCANRawData (hConn, iAxisRef, &stSendRawData_in, &stSendRawData_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.20. MMC_SendCmd

Not in operation

Sends a command string to the drive.

```
MMC_LIB_API int MMC_SendCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_SENDCMD_IN* pInParam,
OUT MMC_SENDCMD_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_SENDCMD_IN** input data structure using the MMC_SendCmd function.

pOutParam

Points to the **MMC_SENDCMD_OUT** output structure receiving information as a result of calling the MMC_SendCmd function.

Remarks

None

Scope

All

MMC_SENDCMD_IN Structure

```
typedef struct{
char pCmd[80];
}MMC_SENDCMD_IN;
```

Parameters

pCmd[80]

Command value. Any +ve or –ve array values with a maximum of 80 characters

MMC_SENDCMD_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SENDCMD_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-43 describes the function block for MMC_SendCmd

Figure 11-43: MMC_SendCmd function block

11.5.20.2. Function Block Code Example

```
int rc;
MMC_SENDCMD_IN     stSendCmd_in;
MMC_SENDCMD_OUT    stSendCmd_out;
//
// Inserting the structure parameters:
strcpy(stSendCmd_in.pCmd, "11111");          //Command value
//
rc = MMC_SendCmd (hConn, iAxisRef, &stSendCmd_in, &stSendCmd_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.21. MMC_SetHeartBeatConsumer

Sets the consumer heartbeat as an event to the user.

```
MMC_LIB_API int MMC_SetHeartBeatConsumerCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SETHEARTBEATCONSUMER_IN* pInParam,
OUT MMC_SETHEARTBEATCONSUMER_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SETHEARTBEATCONSUMER_IN** input data structure using the MMC_SetHeartBeatConsumer function.

pOutParam

Points to the **MMC_SETHEARTBEATCONSUMER_OUT** output structure receiving information as a result of calling the MMC_SetHeartBeatConsumer function.

Remarks

The heartbeat process allows the G-MAS (CAN slave) to monitor the servo drives and to determine if they are active, according to their periodic heartbeat messages. The G-MAS monitors the heartbeat meassge from the servo drives, which may wish to monitor if this slave is active. The consumer heartbeat time defines the expected heartbeat cycle time. Monitoring of the heartbeat starts after the reception of the first heartbeat. If the consumer heartbeat time is 0, the corresponding entry is not used.

If the servo drive does not send a heartbeat message to the G-MAS within the defined period of time (which this API is capable of changing), then a Heartbeat Error Event is sent to the Host Server after a period greater than the Heartbeat message period between the G-MAS and the servo drive.

Scope

Heartbeat mechanism for the CANbus is automatically started when the G-MAS is powered ON, and is set according to the Resource file basic cycle time. A Heartbeat Event is automatically sent to the host system when the heartbeat is delayed by a set time preconfigured in the G-MAS.

The Heartbeat mechanism is set per axis in the Resource file.

MMC_SETHEARTBEATCONSUMER_IN Structure

```
typedef struct{
unsigned int uiHeartbeatTimeFactor;
}MMC_SETHEARTBEATCONSUMER_IN;
```

Parameters

uiHeartbeatTimeFactor

Heart beat time factor is a multiple of 1 ms. The calculation of the basic cycle time (predetermined in the Resource file), multiplied by this heartbeat time factor, and 1 ms, will set the Heartbeat time.

Values accepted are:

0, not in use

>0, any +ve value

MMC_SETHEARTBEATCONSUMER_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SETHEARTBEATCONSUMER_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-44 describes the function block for MMC_SetHeartBeatConsumer

Figure 11-44: MMC_SetHeartBeatConsumer function block

11.5.21.2. Function Block Code Example

```
int rc;
MMC_SETHEARTBEATCONSUMER_IN     stSetHrtBtConsumer_in;
MMC_SETHEARTBEATCONSUMER_OUT    stSetHrtBtConsumer_out;
//
// Inserting the structure parameters:
stSetHrtBtConsumer_in.uiHeartbeatTimeFactor  = 216;   //Heart beat factor per cycle time
//
rc = MMC_SetHeartBeatConsumer (hConn, iAxisRef, &stSetHrtBtConsumer_in,
&stSetHrtBtConsumer_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.22. MMC_SetSyncTime

Where CANbus communication is relevant, sets the Sync time in the communication module, and updates the relevant nodes whose motion mode has an IP address. It also updates the kernel with the Sync time.

```
MMC_LIB_API int MMC_SetSyncTimeCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SETSYNCTIME_IN* pInParam,
OUT MMC_SETSYNCTIME_OUT* pOutParam
);
```

Motion Mode	NC – Not Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SETSYNCTIME_IN** input data structure using the MMC_SetSyncTime function.

pOutParam

Points to the **MMC_SETSYNCTIME_OUT** output structure receiving information as a result of calling the MMC_SetSyncTime function.

Remarks

None

Scope

For use only with distributed systems

MMC_SETSYNCTIME_IN Structure

```
typedef struct{
unsigned short usSYNCTime;
}MMC_SETSYNCTIME_IN;
```

Parameters

usSYNCTime

Sync time input. Refer to the explanation in section **11.4.6 SYNC and Time Stamp on page 11-741**. Accepts any +ve integer value between 0 to 1000.

MMC_SETSYNCTIME_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SETSYNCTIME_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-45 describes the function block for MMC_SetSyncTime

Figure 11-45: MMC_SetSyncTime function block

11.5.22.2. Function Block Code Example

```
int rc;
MMC_SETSYNCTIME_IN          stSetSyncTime_in;
MMC_SETSYNCTIME_OUT         stSetSyncTime_out;
//
// Inserting the structure parameters:
stSetSyncTime_in.usSYNCTime     = 65534; //Sync time input
//
rc = MMC_SetSyncTimeCmd (hConn, &stSetSyncTime_in, &stSetSyncTime_out);
if (rc != 0)
{
    HandleError();
}
```

11.5.23. MMC_StartBulkUpload

The G-MAS manages the bulk upload process upon request from a host, i.e. the host sends this function command to the G-MAS, and the G-MAS uploads the recording buffer.

```
int MMC_StartBulkUploadCmd(IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_STARTBULKUPLOAD_IN* pInParam,
OUT MMC_STARTBULKUPLOAD_OUT* pOutParam
);
```

Motion Mode	NC –Not Supported	Distributed – Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_STARTBULKUPLOAD_IN** input data structure using the MMC_StartBulkUpload function.

pOutParam

Points to the **MMC_STARTBULKUPLOAD_OUT** output structure receiving information as a result of calling the MMC_StartBulkUpload function.

Remarks

None

Scope

For use only with distributed systems.

The parameters usIndex, ucSubIndex, describe a CAN object that can be uploaded. However, not all CAN objects can be uploaded with bulk upload.

MMC_STARTBULKUPLOAD_IN Structure

```
typedef struct mmc_startbulkupload_in{
unsigned short usIndex;
unsigned char ucSubIndex;
}MMC_STARTBULKUPLOAD_IN;
```

Parameters

usIndex

COB index. Any +ve integer values (2 bytes).

ucSubIndex

Defines which index value signifies the group of events to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. From events group 7 and above, the values represent the RPDO output. This parameter should mirror the enumerator value of the ucEventGroup variable, where applicable.

Any +ve character values.

MMC_STARTBULKUPLOAD_OUT Structure

```
typedef struct mmc_startbulkupload_out{
unsigned short usStatus;
short usErrorID;
} MMC_STARTBULKUPLOAD_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-46 describes the function block for MMC_StartBulkUpload.

Figure 11-46: MMC_StartBulkUpload function block

11.5.23.2. Function Code Example

11.5.24. MMC_GetBulkUploadStatus

During the whole process of a bulk upload, the status is retrieved using this function command.

```
int MMC_GetBulkUploadStatusCmd(IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GETBULKUPLOADSTATUS_IN* pInParam,
OUT MMC_GETBULKUPLOADSTATUS_OUT* pOutParam
);
```

Motion Mode	NC –Supported	Distributed – Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETBULKUPLOADSTATUS_IN** input data structure using the MMC_GetBulkUploadStatus function.

pOutParam

Points to the **MMC_GETBULKUPLOADSTATUS_OUT** output structure receiving information as a result of calling the MMC_GetBulkUploadStatus function.

Remarks

This function returns the user with the following data:

- Amount of data received
- Upload state (init/in progress/error/etc.)
- Communication error (if any)
- Process error (if any)

Scope

For use only with distributed systems

MMC_GETBULKUPLOADSTATUS_IN Structure

```
typedef struct mmc_getbulkuploadstatus_in{
unsigned char ucDummy;
}MMC_GETBULKUPLOADSTATUS_IN;
```

Parameters

ucDummy

Dummy value. Any -ve or +ve character.

MMC_GETBULKUPLOADSTATUS_OUT Structure

```
typedef struct mmc_getbulkuploadstatus_out{
unsigned int uiSizeCompleted;
unsigned long ulSizedReported
unsigned short usStatus;
short usErrorID;
short usCommError;
short usUploadError;
unsigned char ucUploadState;
} MMC_GETBULKUPLOADSTATUS_OUT;
```

Parameters

uiSizeCompleted

Number of bytes already received.

ulSizedReported

When the upload starts, the drive reports how many bytes it is transmitting.

usCommError

Communication error. If 0 then upload status is OK. If <0, then the parameter ucUploadState will also produce an error. Refer to the errors listed in sections **6.2 G-MAS Error IDs** and **6.3 Continued G-MAS Error IDs.**

usUploadError

DS301 error. If 0 then upload status is OK. If <0, then refer to the errors listed in sections **6.2 G-MAS Error IDs** and **6.3 Continued G-MAS Error IDs.**

ucUploadState

Upload phase. This may be one of the following:

- Not started
- Pre-initiated
- Initiated
- In progress
- Completed OK
- Completed with failure

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, **6.3 Continued G-MAS Error IDs** and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Displays an error code as -ve or +ve integers.

Figure 11-47 describes the function block for MMC_GetBulkUploadStatus

Figure 11-47: MMC_GetBulkUploadStatus function block

11.5.24.2. Function Block Code Example

11.5.25. MMC_GetBulkUploadData

The G-MAS manages the upload process upon request from a host, i.e. the host sends a "begin upload" command, and the G-MAS uploads the recording buffer. Afterwards, the host sends this function command to retrieve the data buffer.

```
int MMC_GetBulkUploadDataCmd(IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GETBULKUPLOADDATA_IN* pInParam,
OUT MMC_GETBULKUPLOADDATA_OUT* pOutParam)
);
```

Motion Mode	NC –Supported	Distributed – Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETBULKUPLOADDATA_IN** input data structure using the MMC_GetBulkUploadData function.

pOutParam

Points to the **MMC_GETBULKUPLOADDATA_OUT** output structure receiving information as a result of calling the MMC_GetBulkUploadData function.

Remarks

None

Scope

For use only with distributed systems?

MMC_GETBULKUPLOADDATA_IN Structure

```
typedef struct mmc_getbulkuploaddata_in{
unsigned short usStartIndex;
unsigned short usEndIndex;
} MMC_GETBULKUPLOADDATA_IN;
```

Parameters

usStartIndex

Start index value of the uploaded data. +ve values accepted

usEndIndex

End index value of the uploaded data. +ve values accepted

MMC_GETBULKUPLOADDATA_OUT Structure

```
typedef struct mmc_getbulkuploaddata_out{
char cDataBuffer[NC_MAX_REC_PACKET_SIZE];
unsigned short usStatus;
unsigned short usErrorID;
} MMC_GETBULKUPLOADDATA_OUT;
```

Parameters

cDataBuffer[NC_MAX_REC_PACKET_SIZE]

cDataBuffer is the received data. It should be noted that it is the host responsibility to format the data according to its requirements (the data is received as a buffer of bytes)

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-48 describes the function block for MMC_GetBulkUploadData

Figure 11-48: MMC_GetBulkUploadData function block

11.5.25.2. Function Block Code Example

11.6. EtherCAT Drive Communication

Ethernet for Control Automation Technology (EtherCAT) is an open high performance Ethernet-based fieldbus system, which uses the family of industrial computer network protocols used for real-time distributed control, now standardized as IEC 61158. It is a highly flexible Ethernet network protocol that runs over a fast real time Master–Slave network.

The EtherCAT communication speed is up to 100 Mbps full duplex and can include a maximum of 65,535 stations in a single network configuration such as Ethernet star, line or tree without using switches.

Figure 11-49 describes a network of EtherCAT slaves in a ring topology. The Master controls the traffic in the network by initiating the transactions.

Figure 11-49: EtherCAT Network Configuration

Usually, a control system requires the following in periodic time intervals:

- **Inputs**
  Latched Sensors Data such as Positions, Velocities, Currents, System Status, IO's etc,
- **Outputs**
  Control Law commands, or Trajectory Information, or Higher Drive Level Commands.

The specific nature of the data transferred via the network depends on the operation mode of the slave drive. The Device Profile describes the application parameters and the functional behavior of the devices including the device class-specific state machines. A common standard for Drive Device Profiles is the DS-402, CANopen Device Profile, and CoE (Can Over EtherCAT).

The EtherCAT protocol is optimized for process data and is transported directly within the standard IEEE 802.3 Ethernet frame. Each Ethernet frame can include several EtherCAT frames, each serving another slave.

EtherCAT network use a processing on the fly, whereby the Ethernet frame is received and processed while the telegram passes through the device. The frames only delay by a fraction of a microsecond in each node. Using EtherCAT, the entire network can be addressed with just one frame.

The data sequence is independent of the physical order of the nodes in the network; addressing can be in any order. Broadcast, multicast and communication between slaves are possible and must be performed by the master device.

The EtherCAT protocol can be inserted into UDP/IP datagrams. This also enables any control with an Ethernet protocol stack to address EtherCAT systems.

Using the Master configuration tool, the Master scans the EtherCAT network and uses the EtherCAT Slave library to compare the slave memory area that includes information about the slave such as Vendor ID, Product Code, and Slave Configuration.

11.6.1. Elmo EtherCAT

The ELMO environment comprises of three levels (**Figure 11-50**):

- EAS EtherCAT configuration tools
- EtherCAT G-MAS master
- Elmo EtherCAT slave drives

Figure 11-50: EtherCAT Environment

11.6.2. Elmo Slave Drives

The following diagram describes the EtherCAT communication of the drive.

Figure 11-51: Layered Communication protocol in EtherCAT

Physical Layer

The Physical layer of the EtherCAT is a 100Mbits/sec Ethernet port over twisted per cable.

Data Link Layer

This supports two mechanisms of data transfer:

- **Process data**
  Allows writing and reading data simultaneously. This mode is used to transfer the Process data objects (PDO). The PDO transfers via SYNC Manager 2 (PDO_Out) and SYNC Manager 3 (PDO_IN)
- **Mailbox**
  The mailbox mechanism assure that the data will reach to the target. The mailbox is used to transfer the SDOs. The SDO transfers via SYNC Manager 0 (MailboxOut) and SYNC Manager 1 (MailboxIn). SDO objects are used for user triggered access. With SDO services, all of the OD's entries can be read or written. The SDO transport works in asynchronous mode only.

The Elmo drive supports the following communication protocols:

CoE (CANopen over EtherCAT)	Defines a standard way to access the CANopen protocol and includes an object dictionary, SDO, PDO and emergency messages.
EoE (Ethernet over EtherCAT)	Fully Ethernet compatible, defines a standard way to exchange or tunnel standard Ethernet frames.
FoE (File over EtherCAT)	Similar to TFTP, enables access to any data structure in the device, and defines a standard way to download and upload firmware and other files.

The **Object Dictionary (OD)** contains parameters, application data and the mapping information between the process data interface and application data (PDO mapping). Its entries can be accessed via the Service Data Object (SDO).

An Object Dictionary is a naming system that provides a unique identifier to each data item or “object” communicated over the CoE protocol. An object is identified by an index, and if a complex object, by a sub-index as well. CoE and EoE protocols require a set of mandatory objects.

Elmo drive supports distributed clock in order to **synchronize** between the Master and Slaves on the EtherCAT network.

11.6.3. EtherCAT with G-MAS

While a single servo drive can run as a stand-alone drive, without the G-MAS, using its inner profiler and filter. In order to perform synchronized multi axis motions in the system (such as circle, line etc..), a real time communication protocol must be used, and all drives must be synchronized to a specific SYNC signal in the system. The EtherCAT communication protocol enables synchronization of all the controllers to the same SYNC signal by updating the drives in the system with the G-MAS's master time. Thus, all drives in the system are synchronized to the master clock, and all generate an interrupt at exactly the same time.

A profiler can run in the G-MAS, on the condition that the axis (axes) is defined as a vector axis (axes). A vector axis may consist of 1 - 16 axes. The Multi Axis Indexer (MAI) is the profiler that runs within the GMAS, which sends (via a high priority interrupt routine) a calculated set point to the axes in the system and can perform vector calculations for up to 16 axes. The profiler EtherCAT and CAN outputs are points that are to be sent to the specific drives belonging to the vector. Therefore, a number of combination options are available:

- 1 x 16 axes (One vector profiler performing profiles for 16 axes),
- 16 x 1 axes (16 profilers for 16 vector axes), or,
- Any combination of M x N axes – as long as M x N < 16

The SYNC interrupt signal to the drives is based on the ET1100 component in the servo drive. The master G-MAS does not receive this signal, but can calculate when the SYNC signal is generated. This is because the master EtherCAT in the GMAS is responsible for updating the SYNC cycle time in the servo drives, and therefore knows when the SYNC is generated. The MAI can operate at varying cycle times, dependent on a number of parameters, such as the:

- Desired response from the system
- Number of axes participating in the MAI. The more axes, the higher the cycle rate

11.6.4. EtherCAT Gateway

Under normal circumstances, the G-MAS task manages the EtherCAT communications and all devices connected via the EtherCAT. In this situation, adding or removing communications to an Input/Output module or servo drive involves programming the specific EtherCAT resource file. This can involve a quite complex procedure. A more sophisticated alternative exits to stop the operation of the G-MAS normal task manager for EtherCAT via the *MMC_EnableEthercatConfigMode* function block, and directly add or remove the Input/Output module or servo drive, automatically updating the EtherCAT resource file during the process. To perform this Gateway process, the EAS application is used. When the process is completed, the function block *MMC_DisableEthercatConfigMode* is run in the G-MAS to return the task management to the G-MAS using the updated EtherCAT resource file.

11.7. EtherCAT and CANbus Function Blocks

The following EtherCAT drive communication function blocks are described, with the exception of MMC_ETHERCAT_DIAGNOSTICS_INFO, which is a structure:

Drive Communication
MMC_DisableEthercatConfigMode
MMC_EnableEthercatConfigMode
MMC_ECATIODisableDIChangedEvent
MMC_ECATIOEnableDIChangedEvent
MMC_GetCommStatistics
MMC_GetCommDiagnostics
MMC_Get ReactorStatistics

MMC_IsEthercatConfigMode
MMC_ECATIOReadDigitalInput
MMC_ECATIOReadAnalogInput
MMC_ResetCommDiagnostics
MMC_ResetCommStatistics
MMC_SendSDO
MMC_ECATIOWriteAnalogOutput
MMC_ECATIOWriteDigitalOutput

11.7.1. MMC_DisableEthercatConfigMode

Enables the G-MAS task manager and disables direct programming of the G-MAS via the Gateway.

```
MMC_LIB_API int MMC_DisableEthercatConfigMode(
IN MMC_CONNECT_HNDL hConn,
(IN MMC_DISABLE_ECATCONFIGMODE_IN* pInParam)
OUT MMC_DISABLE_ECATCONFIGMODE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_DISABLE_ECATCONFIGMODE_IN** input data structure using the MMC_DisableEthercatConfigMode function.

pOutParam

Points to the **MMC_DISABLE_ECATCONFIGMODE_OUT** output structure receiving information, as a result of calling the MMC_DisableEthercatConfigMode function.

Remarks

Extension of the CANopen technology disabling the Gateway communication between a host system and the G-MAS.

Scope

All

MMC_DISABLE_ECATCONFIGMODE_IN Structure

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

Parameters

ucDummy

Dummy value. Any -ve or +ve character.

MMC_DISABLE_ECATCONFIGMODE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_DISABLE_ECATCONFIGMODE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-52 describes the function block for DisableEthercatConfigMode.

Figure 11-52: DisableEthercatConfigMode function block

11.7.1.2. Function Block Code Example

```
int rc;
MMC_DISABLE_ECATCONFIGMODE_IN       stDisableEcatConfigMode_in;
MMC_DISABLE_ECATCONFIGMODE_OUT      stDisableEcatConfigMode_out;
//
// Inserting the structure parameters:
stDisableEcatConfigMode_in.ucDummy      = 1;     // Dummy input
//
rc = MMC_DisableEthercatConfigMode (hConn, &stDisableEcatConfigMode_out);
if (rc != 0)
{
    HandleError();
}
```

11.7.2. MMC_EnableEthercatConfigMode

Disables the G-MAS task manager to enable direct programming of the G-MAS via the Gateway.

```
MMC_LIB_API int MMC_EnableEthercatConfigMode(
IN MMC_CONNECT_HNDL hConn,
OUT MMC_ENABLE_ECATCONFIGMODE_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pOutParam

Points to the **MMC_ENABLE_ECATCONFIGMODE_OUT** output structure receiving information, as a result of calling the MMC_EnableEthercatConfigMode function.

Remarks

Extension of the CANopen technology enabling the Gateway communication between a host system and the G-MAS.

Scope

All

MMC_ENABLE_ECATCONFIGMODE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_ENABLE_ECATCONFIGMODE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Displays an error code as -ve or +ve integers.

Figure 11-53 describes the function block for MMC_EnableEthercatConfigMode

Figure 11-53: MMC_EnableEthercatConfigMode function block

11.7.2.2. Function Block Code Example

```
int rc;
MMC_ENABLE_ECATCONFIGMODE_IN        stEnableEcatConfigMode_in;
MMC_ENABLE_ECATCONFIGMODE_OUT       stEnableEcatConfigMode_out;
//
// Inserting the structure parameters:
stEnableEcatConfigMode_in.ucDummy        = 1;      // Dummy input
//
rc = MMC_EnableEthercatConfigMode (hConn, &stEnableEcatConfigMode_out);
if (rc != 0)
{
    HandleError();
}
```

11.7.3. MMC_ECATIODisableDIChangedEvent

Disables an EtherCAT I/O input event change against an I/O module.

```
MMC_LIB_API int MMC_ECATIODisableDIChangedEvent (
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_DISABLEDICHANGEDEVENT_IN* pInParam,
OUT MMC_DISABLEDICHANGEDEVENT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_ECATIO_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_DISABLEDICHANGEDEVENT_IN** input data structure using the MMC_ECATIODisableDIChangedEvent function.

pOutParam

Points to the **MMC_DISABLEDICHANGEDEVENT_OUT** output structure receiving information, as a result of calling the MMC_ECATIODisableDIChangedEvent function.

Remarks

When enabled, any EtherCAT I/O input event change is sent from the I/O module to the G-MAS and then host server (if connected).

Scope

All

MMC_DISABLEDICHANGEDEVENT_IN Structure

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

Parameters

ucDummy

Dummy data input. Any +ve character value.

MMC_DISABLEDICHANGEDEVENT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_DISABLEDICHANGEDEVENT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 11-54 describes the function block for MMC_ECATIODisableDIChangedEvent

Figure 11-54: MMC_ECATIODisableDIChangedEvent function block

11.7.3.2. Function Block Code Example

```
int rc;
MMC_DISABLEDICHANGEDEVENT_IN        stDisableDIChangeEv_in;
MMC_DISABLEDICHANGEDEVENT_OUT       stDisableDIChangeEv_out;
//
// Inserting the structure parameters:
stDisableDIChangeEv_in.ucDummy  = 1;     //Dummy data input
//
rc = MMC_ECATIODisableDIChangedEvent (hConn, iAxisRef, &stDisableDIChangeEv_in,
&stDisableDIChangeEv_out);
if (rc != 0)
{
    HandleError();
}
```

11.7.4. MMC_ECATIOEnableDIChangedEvent

Enables an EtherCAT I/O input event change.

```
MMC_LIB_API int MMC_EnableDS401DIChangedEvent(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_ENABLEDICHANGEDEVENT_IN* pInParam,
OUT MMC_ENABLEDICHANGEDEVENT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_ECATIO_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_ENABLEDICHANGEDEVENT_IN** input data structure using the MMC_ECATIOEnableDIChangedEvent function.

pOutParam

Points to the **MMC_ENABLEDICHANGEDEVENT_OUT** output structure receiving information, as a result of calling the MMC_ECATIOEnableDIChangedEvent function.

Remarks

When enabled, any EtherCAT I/O input event change is sent from the I/O module to the G-MAS and then host server (if connected).

Scope

All

MMC_ENABLEDICHANGEDEVENT_IN Structure

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

Parameters

ucDummy

Dummy data input. Any +ve character value.

MMC_ENABLEDICHANGEDEVENT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_ENABLEDICHANGEDEVENT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 11-55 describes the function block for MMC_ECATIOEnableDIChangedEvent

Figure 11-55: MMC_ECATIOEnableDIChangedEvent function block

11.7.4.2. Function Block Code Example

```
int rc;
MMC_ENABLEDICHANGEDEVENT_IN         stEnableDIChangeEv_in;
MMC_ENABLEDICHANGEDEVENT_OUT        stEnableDIChangeEv_out;
//
// Inserting the structure parameters:
stEnableDIChangeEv_in.ucDummy   = 1;     //Dummy data input
//
rc = MMC_ECATIOEnableDIChangedEvent (hConn, iAxisRef, &stEnableDIChangeEv_in,
&stEnableDIChangeEv_out);
if (rc != 0)
{
    HandleError();
}
```

11.7.5. MMC_ECATIOReadDigitalInput

Reads the EtherCAT I/O input of all 64 bit I/O's in one action, increasing the communication speed proportionately versus reading 8 x groups of 8 I/O's.

```
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
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_ECATIO_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READDI_IN** input data structure using the MMC_ECATIOReadDigitalInput function.

pOutParam

Points to the **MMC_READDI_OUT** output structure receiving information, as a result of calling the MMC_ECATIOReadDigitalInput function.

Remarks

None

Scope

All

MMC_READDI_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_READDI_IN;
```

Parameters

dummy

Dummy data input. Any +ve character value.

MMC_READDI_OUT Structure

```
typedef struct{
#ifdef WIN32
unsigned __int64 ulliDI;
#else
unsigned long long int ulliDI;
#endif
unsigned short usStatus;
short usErrorID;
}MMC_READDI_OUT;
```

Parameters

__int64 ulliDI or ulliDI

If function is defined for WIN32 then use *__int64 ulliDI*, else use *ulliDI*. Any +ve, –ve (Win32) or +ve 64bit (8 bytes) character and/or integer.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-56 describes the function block for MMC_ECATIOReadDigitalInput

Figure 11-56: MMC_ECATIOReadDigitalInput function block

11.7.5.2. Function Block Code Example

```
int rc;
MMC_READDI_IN      stReadDI_in;
MMC_READDI_OUT     stReadDI_out;
//
// Inserting the structure parameters:
stReadDI_in.dummy      = 1;      //dummy input
//
rc = MMC_ECATIOReadDigitalInput (hConn, iAxisRef, &stReadDI_in, &stReadDI_out);
printf("EtherCAT Input Status[%ld] ErrId[%d]\n", (long int)stReadDI_out.ulliDI,
(short)stReadDI_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.7.6. MMC_ECATIOReadAnalogInput

Reads the EtherCAT I/O analog input.

```
MMC_LIB_API int MMC_ECATIOReadAnalogInput(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READAI_IN* pInParam,
OUT MMC_READAI_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_ECATIO_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READAI_IN** input data structure using the MMC_ECATIOReadAnalogInput function.

pOutParam

Points to the **MMC_READAI_OUT** output structure receiving information, as a result of calling the MMC_ECATIOReadAnalogInput function.

Remarks

None

Scope

All

MMC_READAI_IN Structure

```
typedef struct mmc_readai_in{
unsigned char ucIndex;
}MMC_READAI_IN;
```

Parameters

ucIndex

Analog input index. Any +ve character value.

MMC_READAI_OUT Structure

```
typedef struct mmc_readai_out{
short sAI;
unsigned short usStatus;
short usErrorID;
}MMC_READAI_OUT;
```

Parameters

sAI

Analog Input value. Any +ve value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 11-57 describes the function block for MMC_ECATIOReadAnalogInput

Figure 11-57: MMC_ECATIOReadAnalogInput function block

11.7.6.2. Function Block Code Example

```
int rc;
MMC_READDI_IN     stReadDI_in;
MMC_READDI_OUT    stReadDI_out;
//
// Inserting the structure parameters:
stReadDI_in.dummy      = 1;     //dummy input
//
rc = MMC_ECATIOReadAnalogInput (hConn, iAxisRef, &stReadDI_in, &stReadDI_out);
printf("EtherCAT Analog Input Status[%ld] ErrId[%d]\n", (long int)stReadDI_out.ulliDI,
(short)stReadDI_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.7.7. MMC_ECATIOWriteAnalogOutput

Writes to the EtherCAT I/O analog outputs.

```
MMC_LIB_API int MMC_ECATIOWriteAnalogOutput(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_WRITEAO_IN* pInParam,
OUT MMC_WRITEAO_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_ECATIO_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEAO_IN** input data structure using the MMC_ECATIOWriteAnalogOutput function.

pOutParam

Points to the **MMC_WRITEAO_OUT** output structure receiving information, as a result of calling the MMC_ECATIOWriteAnalogOutput function.

Remarks

None

Scope

All

MMC_WRITEAO_IN Structure

```
typedef struct mmc_writeao_in{
short sAO;
unsigned char ucIndex;
}MMC_WRITEAO_IN;
```

Parameters

sAO

Analog Output value. Any +ve value.

ucIndex

Analog input index. Any +ve character value.

MMC_WRITEAO_OUT Structure

```
typedef struct mmc_writeao_out{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEAO_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-58 describes the function block for MMC_ECATIOWriteAnalogOutput

Figure 11-58: MMC_ECATIOWriteAnalogOutput function block

11.7.7.2. Function Block Code Example

```
int rc;
MMC_WRITEDO_IN          stWriteDO_in;
MMC_WRITEDO_OUT         stWriteDO_out;
//
// Inserting the structure parameters:
stWriteDO_in.ulliDO    = 1;     //Index of the group axes
//
rc = MMC_ECATIOWriteDigitalOutput (hConn, iAxisRef, &stWriteDO_in, &stWriteDO_out);
if (rc != 0)
{
    HandleError();
}
```

11.7.8. MMC_ECATIOWriteDigitalOutput

Writes to the EtherCAT I/O outputs of all 64 bit I/O's in one action, increasing the communication speed proportionately versus writing to 8 x groups of 8 I/O's.

```
MMC_LIB_API int MMC_ECATIOWriteDigitalOutput(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_WRITEDO_IN* pInParam,
OUT MMC_WRITEDO_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_ECATIO_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEDO_IN** input data structure using the MMC_ECATIOWriteDigitalOutput function.

pOutParam

Points to the **MMC_WRITEDO_OUT** output structure receiving information, as a result of calling the MMC_ECATIOWriteDigitalOutput function.

Remarks

None

Scope

All

MMC_WRITEDO_IN Structure

```
typedef struct{
#ifdef WIN32
unsigned __int64 ulliDO;
#else
unsigned long long int ulliDO;
#endif
}MMC_WRITEDO_IN;
```

Parameters

__int64 ulliDI or ulliDI

If function is defined for WIN32 then use *__int64 ulliDI*, else use *ulliDI*. Any +ve, –ve (Win32) or +ve 64bit (8 bytes) character and/or integer.

MMC_WRITEDO_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEDO_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-59 describes the function block for MMC_ECATIOWriteDigitalOutput

Figure 11-59: MMC_ECATIOWriteDigitalOutput function block

11.7.8.2. Function Block Code Example

```
int rc;
MMC_WRITEDO_IN          stWriteDO_in;
MMC_WRITEDO_OUT         stWriteDO_out;
//
// Inserting the structure parameters:
stWriteDO_in.ulliDO    = 1;     //Index of the group axes
//
rc = MMC_ECATIOWriteDigitalOutput (hConn, iAxisRef, &stWriteDO_in, &stWriteDO_out);
if (rc != 0)
{
    HandleError();
}
```

11.7.9. MMC_GetCommStatistics

Receives communication statistics for a specific axis.

It is recommended to use the function **MMC_GetEthercatCommStatistics on page 11-855** rather than this function. The GetCommStatistics function is for legacy applications only.

```
MMC_LIB_API int MMC_GetCommStatistics(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GETCOMMSTATISTICS_IN* pInParam,
OUT MMC_GETCOMMSTATISTICS_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GETCOMMSTATISTICS_IN** input data structure using the MMC_GetCommStatistics function.

pOutParam

Points to the **MMC_GETCOMMSTATISTICS_OUT** output structure receiving information as a result of calling the MMC_GetCommStatistics function.

Remarks

None

Scope

All

MMC_GETCOMMSTATISTICS_IN Structure

```
typedef struct{
unsigned short usAxesRef        /*usSlaveID*/;
}MMC_GETCOMMSTATISTICS_IN;
```

Parameters

usAxesRef

Axis/group reference handle type returned by GetAxisRef command

MMC_GETCOMMSTATISTICS_OUT Structure

```
typedef struct{
unsigned long dwSendErrors;
unsigned long dwReceiveErrors;
unsigned long dwWrongWC;
unsigned long dwParseErrors;
unsigned short usNumOfSlaves;
unsigned short usStatus;
short usErrorID;
unsigned char ucMasterState;
unsigned char ucSlaveState;
}MMC_GETCOMMSTATISTICS_OUT;
```

Parameters

dwSendErrors

Send errors counter. Any +ve 32 bit value.

dwReceiveErrors

Receive errors counter. Any +ve 32 bit value.

dwWrongWC

Wrong writer counter number. Any +ve 32 bit value

dwParseErrors

Parse error counter. Any +ve 32 bit value.

usNumOfSlaves

Checks the realistic number of drives connected to the G-MAS. Limited +ve integer value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

ucMasterState

G-MAS specification. +ve character value

ucSlaveState

Specification state of the slaves.

Figure 11-60 describes the function block for MMC_GetCommStatistics

Figure 11-60: MMC_GetCommStatistics function block

11.7.9.2. Function Block Code Example

```
int rc;
MMC_GETCOMMSTATISTICS_IN        stGetCOMMStats_in;
MMC_GETCOMMSTATISTICS_OUT       stGetCOMMStats_out;
//
// Inserting the structure parameters:
stGetCOMMStats_in.usAxesRef     = 19;    //Axis reference
//
rc = MMC_GetCommStatistics (hConn, iAxisRef, &stGetCOMMStats_out);
printf("Comm Statistics Status[%ld][%ld][%ld][%ld] ErrId[%d]\n", (long
int)stGetCOMMStats_out.dwSendErrors, (long int)stGetCOMMStats_out.dwReceiveErrors, (long
int)stGetCOMMStats_out.dwWrongWC, (long int)stGetCOMMStats_out.dwParseErrors,
(short)stGetCOMMStats_out.usErrorID);
printf("Comm Statistics Status[%ld][%ld][%ld]\n", (long int)stGetCOMMStats_out.usNumOfSlaves,
(long int)stGetCOMMStats_out.ucMasterState, (long int)stGetCOMMStats_out.ucSlaveState);
if (rc != 0)
{
    HandleError();
}
```

11.7.10. MMC_GetEthercatCommStatistics

Obtains the EtherCAT communication statistics used as part of the FoE download mechanism in the EAS application.

Important: **To use this function refer to Elmo for support.**

```
MMC_LIB_API int MMC_GetEthercatCommStatistics(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETCOMMSTATISTICSEX_IN* pInParam,
OUT MMC_GETCOMMSTATISTICSEX_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETCOMMSTATISTICSEX_IN** input data structure using the MMC_GetEthercatCommStatistics function.

pOutParam

Points to the **MMC_GETCOMMSTATISTICSEX_OUT** output structure receiving information, as a result of calling the MMC_GetEthercatCommStatistics function.

Remarks

A maximum number of 76 slaves are allowed to be attached for downloading using this single function.

Scope

EtherCAT

MMC_GETCOMMSTATISTICSEX_IN Structure

```
typedef struct mmc_getcommstatisticsex_in{
unsigned short pwSlaveId[ETHERCAT_STATISTICSEX_MAX_SLAVES];
unsigned char ucSlavesNum;
}MMC_GETCOMMSTATISTICSEX_IN;
```

Parameters

pwSlaveId[ETHERCAT_STATISTICSEX_MAX_SLAVES]

The array of slave ID that the user wishes to view with a maximum number of 76 slaves. The user enters each slave ID from 0 to to the (maximum number of slave supported-1).

ucSlavesNum

The number of slaves that the user wishes to view attached to the master G-MAS. Values are +ve characters

MMC_GETCOMMSTATISTICSEX_OUT Structure

```
typedef struct mmc_getcommstatisticsex_out{
unsigned long dwSendErrors;
unsigned long dwReceiveErrors;
unsigned long dwWrongWC;
unsigned long dwParseErrors;
MMC_ECAT_SII_CONTENT pstSII_Content[ETHERCAT_STATISTICSEX_MAX_SLAVES];
unsigned short usNumOfSlaves;
unsigned short usStatus;
unsigned short usErrorID;
unsigned char ucMasterState;
unsigned char pucAxesState[ETHERCAT_STATISTICSEX_MAX_SLAVES];
unsigned char pucAxesDiagnosticState[ETHERCAT_STATISTICSEX_MAX_SLAVES];
unsigned char ucMasterDiagnosticState;
}MMC_GETCOMMSTATISTICSEX_OUT;
```

Parameters

dwSendErrors

The number of EtherCAT transmission errors. Any +ve value is recieved.

dwReceiveErrors

The number of EtherCAT recieve errors. Any +ve value is recieved.

dwWrongWC

The value of the incorrect working counter. Any +ve value is recieved.

dwParseErrors

The number of Parse errors encountered. Any +ve value is recieved.

MMC_ECAT_SII_CONTENT pstSII_Content[ETHERCAT_STATISTICSEX_MAX_SLAVES]

The details of the drive connected to the Master, the ECAT System Information in an array of system information with a maximum number of 76 slaves.

```
typedef struct _MMC_ECAT_SII_CONTENT{
unsigned long ulVendorId;
unsigned long ulProductCode;
unsigned long ulRevisionNo;
unsigned long ulSerialNo;
} MMC_ECAT_SII_CONTENT;
```

ulVendorId

The vendor ID of the slave drive. Any +ve value is recieved.

ulProductCode

The product code for the slave drive. Any +ve value is recieved.

ulRevisionNo

The revision number of the save drive. Any +ve value is recieved.

ulSerialNo

The serial number of the slave drive. Any +ve value is recieved.

usNumOfSlaves

The number of slaves attached to the master G-MAS. Values are +ve numbers

ucMasterState

The state of the master according to the following definition:

EcatStateNotSet	= 0x00	State is not set
EcatStateI	= 0x01	EtherCAT State "Init"
EcatStateP	= 0x02	EtherCAT State "Pre-Operational”
EcatStateB	= 0x03	EtherCAT State "Bootstrap”
EcatStateS	= 0x04	EtherCAT State "Safe-Operational”
EcatStateO	= 0x08	EtherCAT State "Operational"

pucAxesState[ETHERCAT_STATISTICSEX_MAX_SLAVES]

The array of slave states with a maximum of 76, each displayed with the state defined by the following:

EcatStateNotSet	= 0x00	State is not set
EcatStateI	= 0x01	EtherCAT State "Init"
EcatStateP	= 0x02	EtherCAT State "Pre-Operational”
EcatStateB	= 0x03	EtherCAT State "Bootstrap”
EcatStateS	= 0x04	EtherCAT State "Safe-Operational”
EcatStateO	= 0x08	EtherCAT State "Operational"

pucAxesDiagnosticState[ETHERCAT_STATISTICSEX_MAX_SLAVES]

The diagnostics of an array of slaves with a maximum number of 76 slaves. This diagnostics has the following values, and explanations:

Slave doesn't respond to commands	EcatSlaveDiagnosticStateOffLine	0x00000001
Slave's state is different as the set one	EcatSlaveDiagnosticStateErrorEcatState	0x00000002
Slave is not configured	EcatSlaveDiagnosticStateNotConfigured	0x00000004
Slave's configuration doesn't match the configuration for this slave found in master	EcatSlaveDiagnosticStateWrongConfiguration	0x00000008
	EcatSlaveDiagnosticStateInitCmdError	0x00000010
	EcatSlaveDiagnosticStateMailboxInitCmdError	0x00000020

ucMasterDiagnosticState

The diagnostics of the Master with the following values and explanations:

Diagnostics completed successfully	EcatMasterDiagnosticStateUpdated	0x00000001
Error while sending/receiving a	EcatMasterDiagnosticStateSendReceiveError	0x00000002

frame		
Error while processing the received frame	EcatMasterDiagnosticStateParseError	0x00000004
No connection between the NIC adapter and slaves	EcatMasterDiagnosticStateLinkDown	0x00000008
Wrong configuration	EcatMasterDiagnosticStateWrongConfiguration	0x00000010
Slave-to-slave timeout	EcatMasterDiagnosticStateS2STimeout	0x00000020
Default data was set	EcatMasterDiagnosticStateDefaultDataWasSet	0x00000040
WatchDogTimeOut	EcatMasterDiagnosticStateWatchDogTimeOut	0x00000080

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-61 describes the function for MMC_GetEthercatCommStatistics

Figure 11-61: MMC_GetEthercatCommStatistics function

11.7.10.2. Function Code Example

```
void DownloadFoe()
{
     MMC_DOWNLOADFOE_IN dlfoe ;
     MMC_DOWNLOADFOE_OUT dlfoeout ;
     MMC_GETFOESTATUS_OUT foestat ;
     MMC_GET_GMASOP_MODE_OUT pOpmode ;

     MMC_GETCOMMSTATISTICSEX_IN gcstat_In ;
     MMC_GETCOMMSTATISTICSEX_OUT gcstat_Out ;
     int i ;
     //
     //
     // Before DownloadingFOE - It is good practice that drives will be reset because
     // if one of the drives is after DownloadFoE and was not reset, its state and statitstics
are unknown.
     //
     dlfoe.pwSlaveId[0]=0 ;      // Note: Slave ID is inserted here !!
     dlfoe.pwSlaveId[1]=1 ;      // Note: Slave ID is inserted here !!
     //
     dlfoe.ucSlavesNum = 2;      // Number of relevant slaves in the pwSlaveId array.
     //
     // Same for slave statistics:
     gcstat_In.pwSlaveId[0] = 0 ;
     gcstat_In.pwSlaveId[1] = 1 ;
```

```
    gcstat_In.ucSlavesNum = 2 ;
    //
    // Insert IP of tftp server. Usually the connection IP of the PC.
    dlfoe.pucServer[0] = 10 ;
    dlfoe.pucServer[1] = 10 ;
    dlfoe.pucServer[2] = 20 ;
    dlfoe.pucServer[3] = 55 ;
    //
    // Copy file path name to the structure. Should be relative to the tftp folder
    strcpy(dlfoe.pcFileName,"FoEFW  01.01.04.68 27Oct2011P01G.abs") ;

    // Start tftp server on host. Only then call the MMC_DownloadFoE.
    //
    int rc = MMC_DownloadFoE(conn_hndl,&dlfoe,&dlfoeout)  ;
    if(rc < 0)
    {
        // Error Calling MMC_DownloadFoE. Error in dlfoeout.usErrorID
        return ;
    }
    //
    // If we reached this line, the tftp was succesful. Poll the GMAS for results:
    while(TRUE)
    {
        Sleep(100) ;
        //
        // Check the FoE progress:
        MMC_GetFoEStatus(conn_hndl,&foestat);
        if(rc < 0)
        {
            // Error Calling MMC_GetFoEStatus. Error in foestat.usErrorID
            return ;
        }
        //
        // Check that the FoE started.
        if(foestat.ucFOEStarted)
        {
            rc = MMC_GetGMASOperationMode(conn_hndl,&pOpmode) ;
            if(rc < 0)
            {
                // Error Calling MMC_GetGMASOperationMode. Error in pOpmode.usErrorID
                return ;
            }
            // Print Remaining time - foestat.ucProgress
            //
            // Check Foe Download progress is over and GMAS back in operational mode.
            if ((foestat.ucProgress == 0) && (pOpmode.ucResult == 0))
            {
                //
                // Download over. Check to see if any drives failed.
                for(i = 0 ; i < dlfoe.ucSlavesNum ; i++)
                {
                    if(foestat.pstSlavesErrorID[i].sErrorID != 0)
                    {
                        // Error on one of the slaves. Print error:
                        // SlaveID: foestat.pstSlavesErrorID[i].usSlaveID  has error -
foestat.pstSlavesErrorID[i]
                    }
                }
                // Notify user to switch drives Off / On and then check the download status.
wait 5 sec's.
                //
                // please note - MAX 76 slaves can be read.
                rc = MMC_GetEthercatCommStatistics(conn_hndl,&gcstat_In,&gcstat_Out) ;
                if(rc < 0)
                {
                    // Error Calling MMC_GetEthercatCommStatistics. Error in
gcstat_Out.usErrorID
                    return ;
                }
```

```
                    //
                    // gcstat_Out.ucMasterState - Should be EcatStateO  . operational.
                    // Good idea to read number of slaves on bus - gcstat_Out.usNumOfSlaves
                    // gcstat_Out.ucMasterDiagnosticState - All bits should be 0, except for:
EcatMasterDiagnosticStateUpdated, EcatMasterDiagnosticStateDefaultDataWasSet bits.
                    //
                    for(i = 0 ; i < gcstat_In.ucSlavesNum ; i++)
                    {
                         // gcstat_Out.pucAxesState[i] - Should be EcatStateO.
                         // gcstat_Out.pucAxesDiagnosticState[i] - Should be 0.
                         if((gcstat_Out.pstSII_Content[i].ulVendorId == 0x9A) &&
(gcstat_Out.pstSII_Content[i].ulRevisionNo <= 0xFF))
                         {
                              //
                              // Drive stuck in no firmware state. Notify User. In This case the
InitCmdFail in diagnostics is not relevant.
                         }
                    }
                    return ;
               }
          }
     }
}

int OnConnectGetDiagnostics()
{
     int rc ;
     MMC_GET_GMASOP_MODE_OUT pOpmode ;

     MMC_GETCOMMSTATISTICSEX_IN gcstat_In ;
     MMC_GETCOMMSTATISTICSEX_OUT gcstat_Out ;
     int i ;
     //
     rc = MMC_GetGMASOperationMode(conn_hndl,&pOpmode) ;
     if(rc < 0)
     {
          // Error Calling MMC_GetGMASOperationMode. Error in pOpmode.usErrorID
          return ;
     }
     //
     // Check GMAS Operational state. If == 2, then in Download FOE state.
     if (pOpmode.ucResult == 2)
     {
          // GMAS in Download FoE state. We decided that a mesage will be shown to user that
the GMAS is in Download FoE.
     }

     rc = MMC_GetEthercatCommStatistics(conn_hndl,&gcstat_In,&gcstat_Out) ;
     if(rc < 0)
     {
          // Error Calling MMC_GetEthercatCommStatistics. Error in gcstat_Out.usErrorID
          return ;
     }
     //
     // gcstat_Out.ucMasterState - Should be EcatStateO. operational.
     // Good idea to read number of slaves on bus - gcstat_Out.usNumOfSlaves. Should be
identical to number of drives configured.
     // gcstat_Out.ucMasterDiagnosticState - All bits should be 0, except for:
EcatMasterDiagnosticStateUpdated, EcatMasterDiagnosticStateDefaultDataWasSet bits.
     //
     for(i = 0 ; i < gcstat_In.ucSlavesNum ; i++)
     {
          // gcstat_Out.pucAxesState[i] - Should be EcatStateO.
          //
          if((gcstat_Out.pstSII_Content[i].ulVendorId == 0x9A) &&
(gcstat_Out.pstSII_Content[i].ulRevisionNo <= 0xFF))
          {
               //
```

```
              // One of the Drives is stuck in no firmware state. Notify User to go to
diagnostics tab - NOT TO CONFIGURATOR.
              // In this case - the gcstat_Out.pucAxesDiagnosticState[i] InitCmd bit may be
set..
          }
          else
          {
              // pucAxesDiagnosticState[i] should be 0 !
          }

     }
}
```

11.7.11. MMC_GetCommDiagnostics

Receives communication diagnostics for specific axis.

```
MMC_LIB_API int MMC_GetCommDiagnostics(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETCOMMDIAGNOSTICS_IN* pInParam,
OUT MMC_GETCOMMDIAGNOSTICS_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETCOMMDIAGNOSTICS_IN** input data structure using the MMC_GetCommDiagnostics function.

pOutParam

Points to the **MMC_GETCOMMDIAGNOSTICS_OUT** output structure receiving information, as a result of calling the MMC_GetCommDiagnostics function.

Remarks

None

Scope

Not limited

MMC_GETCOMMDIAGNOSTICS_IN Structure

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

Parameters

ucDummy

Dummy data input. Any +ve character value.

MMC_GETCOMMDIAGNOSTICS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
MMC_ETHERCAT_DIAGNOSTICS_INFO pDiagnosticsSlavesArr[ETHERCAT_ID_MAX];
}MMC_GETCOMMDIAGNOSTICS_OUT;
```

Parameters

pDiagnosticsSlavesArr

pDiagnosticsSlavesArr is an enumerator and ID integer ranging between [1 – 100] describing the maximum number of all EtherCAT connected systems, including the G-MAS.

Where [ETHERCAT_ID_MAX] is an ID with values [1 – 100] array of all slaves attached to master bus.

MMC_ETHERCAT_DIAGNOSTICS_INFO defines the EtherCAT diagnostics info. This structure physically checks the connection of the four connection ports of the G-MAS, and checks for three possible types of errors.

ucRXErrorsPort0

Receiving errors at the Port 0. Any +ve character

ucInvalidFramesPort0

Invalid frames at the Port 0. Any +ve character

ucLostLinkErrorsPort0

Lost link errors at the Port 0. Any +ve character

ucRXErrorsPort1

Receiving errors at the Port 1. Any +ve character

ucInvalidFramesPort1

Invalid frames at the Port 1. Any +ve character

ucLostLinkErrorsPort1

Lost link errors at the Port 1. Any +ve character

ucRXErrorsPort2

Receiving errors at the Port 2. Any +ve character

ucInvalidFramesPort2

Invalid frames at the Port 2. Any +ve character

ucLostLinkErrorsPort2

Lost link errors at the Port 2. Any +ve character

ucRXErrorsPort3

Receiving errors at the Port 3. Any +ve character

ucInvalidFramesPort3

Invalid frames at the Port 3. Any +ve character

ucLostLinkErrorsPort3

Lost link errors at the Port 3. Any +ve character

Figure 11-62 describes the function block for MMC_GetCommDiagnostics

Figure 11-62: MMC_GetCommDiagnostics function block

11.7.11.2. Function Block Code Example

```
int rc;
MMC_GETCOMMDIAGNOSTICS_IN       stGetCOMMDiags_in;
MMC_GETCOMMDIAGNOSTICS_OUT      stGetCOMMDiags_out;
//
// Inserting the structure parameters:
stGetCOMMDiags_in.ucDummy  = 1; //Dummy data input
//
rc = MMC_GetCommDiagnostics (hConn, &stGetCOMMDiags_in, &stGetCOMMDiags_out);
printf("Comm Diagnostics Status[%ld] ErrId[%d]\n", (long
int)stGetCOMMDiags_out.pDiagnosticsSlavesArr, (short)stGetCOMMDiags_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.7.12. MMC_GetReactorStatistics

Obtains the statistics from the G-MAS server base processor.

```
MMC_LIB_API int MMC_GetReactorStatistics(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_GETREACTORSTATISTICS_IN* pInParam,
OUT MMC_GETREACTORSTATISTICS_OUT* pOutParam
);
```

Motion Mode	NC – Supported	Distributed –Supported
Source	C:\GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_GETREACTORSTATISTICS_IN** input data structure using the MMC_GetReactorStatistics function.

pOutParam

Points to the **MMC_GETREACTORSTATISTICS_OUT** output structure receiving information as a result of calling the MMC_GetReactorStatistics function.

Remarks

This base processor is responsible for organizing the PDO's and performing their respective operations in their relevant order. This function block counts the number of PDO's sent to the base processor and issues a report of the total count.

Scope

All

MMC_GETREACTORSTATISTICS_IN Structure

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

Parameters

ucDummy

Dummy input data. Any +ve character value.

MMC_GETREACTORSTATISTICS_OUT Structure

```
typedef struct{
int iReactorQueueSize;
unsigned short usStatus;
unsigned short usErrorID;
}MMC_GETREACTORSTATISTICS_OUT;
```

Parameters

iReactorQueueSize

Returned reactor queue size inquiry integer. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-63 describes the function block for MMC_GetReactorStatistics

Figure 11-63: MMC_GetReactorStatistics function block

11.7.12.2. Function Block Code Example

```
int rc;
MMC_GETREACTORSTATISTICS_IN         stGetReactorStats_in;
MMC_GETREACTORSTATISTICS_OUT        stGetReactorStats_out;
//
// Inserting the structure parameters:
stGetReactorStats_in.ucDummy    = 1; //Dummy data input
//
rc = MMC_GetReactorStatistics (hConn, iAxisRef, &stGetReactorStats_in,
&stGetReactorStats_out);
printf("Reactor Statistics[%ld] ErrId[%d]\n", (long
int)stGetReactorStats_out.iReactorQueueSize, (short)stGetReactorStats_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.7.13. MMC_IsEthercatConfigMode

Defines whether the EtherCAT configuration mode is operational or not.

```
MMC_LIB_API int MMC_IsEthercatConfigMode(
IN MMC_CONNECT_HNDL hConn,
IN MMC_IS_ECATCONFIGMODE_IN* pInParam
OUT MMC_IS_ECATCONFIGMODE_OUT* pOutParam
);
```

Motion Mode	NC – Supported	Distributed - Supported
Source	C:\GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_IS_ECATCONFIGMODE_IN** input data structure using the MMC_IsEthercatConfigMode function.

pOutParam

Points to the **MMC_IS_ECATCONFIGMODE_OUT** output structure receiving information as a result of calling the MMC_IsEthercatConfigMode function.

Remarks

Extension of the CANopen technology questioning the Gateway communication between a host system and the G-MAS.

Scope

All

MMC_IS_ECATCONFIGMODE_IN Structure

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

Parameters

ucDummy

Dummy input data. Any +ve character value.

MMC_IS_ECATCONFIGMODE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned char ucResult;
}MMC_IS_ECATCONFIGMODE_OUT;
```

Parameters

ucResult

Returned answer to question "Is the EtherCAT Config mode operational?" Yes or No.

Character but Boolean, 0 or 1

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-64 describes the function block for MMC_IsEthercatConfigMode

Figure 11-64: MMC_IsEthercatConfigMode function block

11.7.13.2. Function Block Code Example

```
int rc;
MMC_IS_ECATCONFIGMODE_IN        stIsEcatConfigMode_in;
MMC_IS_ECATCONFIGMODE_OUT       stIsEcatConfigMode_out;
//
// Inserting the structure parameters:
stIsEcatConfigMode_in.ucDummy       = 1;     // Dummy input
//
rc = MMC_IsEthercatConfigMode (hConn, &stIsEcatConfigMode_out);
printf("Is EtherCAT Config Mode Status[%ld] ErrId[%d]\n", (long
int)stIsEcatConfigMode_out.ucResult, (short)stIsEcatConfigMode_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.7.14. MMC_ResetCommDiagnostics

Resets the CRC counters registers of all slaves on the bus to 0. The CRC counters registers can be retrieved via the *GetCommDiagnostics* function.

```
MMC_LIB_API int MMC_ResetCommDiagnostics(
IN MMC_CONNECT_HNDL hConn,
IN MMC_RESETCOMMDIAGNOSTICS_IN* pInParam,
OUT MMC_RESETCOMMDIAGNOSTICS_OUT* pOutParam
);
```

Motion Mode	NC – Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_RESETCOMMDIAGNOSTICS_IN** input data structure using the MMC_ResetCommDiagnostics function.

pOutParam

Points to the **MMC_RESETCOMMDIAGNOSTICS_OUT** output structure receiving information as a result of calling the MMC_ResetCommDiagnostics function.

Remarks

None

Scope

All

MMC_RESETCOMMDIAGNOSTICS_IN Structure

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

Parameters

ucDummy

Dummy input data. Any +ve character value.

MMC_RESETCOMMDIAGNOSTICS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_RESETCOMMDIAGNOSTICS_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-65 describes the function block for MMC_ResetCommDiagnostics

Figure 11-65: MMC_ResetCommDiagnostics function block

11.7.14.2. Function Block Code Example

```
int rc;
MMC_RESETCOMMDIAGNOSTICS_IN         stResetCOMMDiags_in;
MMC_RESETCOMMDIAGNOSTICS_OUT        stResetCOMMDiags_out;
//
// Inserting the structure parameters:
stResetCOMMDiags_in.ucDummy     = 1; //Dummy data input
//
rc = MMC_ResetCommDiagnostics (hConn, &stResetCOMMDiags_in, &stResetCOMMDiags_out);
if (rc != 0)
{
    HandleError();
}
```

11.7.15. MMC_ResetCommStatistics

Reset all communication statistics. Resets the communication error counters.

```
MMC_LIB_API int MMC_ResetCommStatistics(
IN MMC_CONNECT_HNDL hConn,
IN MMC_RESETCOMMSTATISTICS_IN* pInParam,
OUT MMC_RESETCOMMSTATISTICS_OUT* pOutParam
);
```

Motion Mode	NC – Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_RESETCOMMSTATISTICS_IN** input data structure using the MMC_ResetCommStatistics function.

pOutParam

Points to the **MMC_RESETCOMMSTATISTICS_OUT** output structure receiving information as a result of calling the MMC_ResetCommStatistics function.

Remarks

None

Scope

All

MMC_RESETCOMMSTATISTICS_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_RESETCOMMSTATISTICS_IN;
```

Parameters

ucDummy

Dummy input data. Any +ve character value.

MMC_RESETCOMMSTATISTICS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_RESETCOMMSTATISTICS_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 11-66 describes the function block for MMC_ResetCommStatistics

Figure 11-66: MMC_ResetCommStatistics function block

11.7.15.2. Function Block Code Example

```
int rc;
MMC_RESETCOMMSTATISTICS_IN      stResetCOMMStats_in;
MMC_RESETCOMMSTATISTICS_OUT     stResetCOMMStats_out;
//
// Inserting the structure parameters:
stResetCOMMStats_in.dummy  = 1; //Dummy data input
//
rc = MMC_ResetCommStatistics (hConn, &stResetCOMMStats_in, &stResetCOMMStats_out);
if (rc != 0)
{
    HandleError();
}
```

11.7.16. MMC_SendSDO

Sends SDO message command, in units of 1, 2, or 4 bytes.

```
MMC_LIB_API int MMC_SendSdoCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_SENDSDO_IN* pInParam,
OUT MMC_SENDSDO_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_SENDSDO_IN** input data structure using the MMC_SendSdo function.

pOutParam

Points to the **MMC_SENDSDO_OUT** output structure receiving information, as a result of calling the MMC_SendSdo function.

Remarks

The data length parameter MMC_SendSDO receives in bytes. However, if a user sends a request to upload/download an SDO object that is not 1, 2 or 4 bytes, an error is returned.

Scope

All

MMC_SENDSDO_IN Structure

```
typedef struct{
long lData;
unsigned long ulDataLength;
unsigned short usSlaveID;
unsigned short usIndex;
unsigned char ucSubIndex;
unsigned char ucService;
}MMC_SENDSDO_IN;
```

Parameters

lData

Data. Unlimited +ve or –ve values (long).

ulDataLength

Length of the data with 1, 2, or 4 values in bytes

usSlaveID

The slave ID. Any +ve integer value. This variable is redundant and no value should be entered.

usIndex

COB index. Any +ve integer values (2 bytes).

ucSubIndex

Defines which index value signifies the group of events to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. From events group 7 and above, the values represent the RPDO output. This parameter should mirror the enumerator value of the ucEventGroup variable, where applicable.

Any +ve character values.

ucService

Defines whether the input is a download or upload. Accepted integer values of:

1 - Download

2 - Upload

MMC_SENDSDO_OUT Structure

```
typedef struct{
long lData;
unsigned long ulDataLength;
unsigned short usStatus;
short usErrorID;
}MMC_SENDSDO_OUT;
```

Parameters

lData

Data. Unlimited +ve or –ve values (long).

ulDataLength

Length of the data. Unlimited +ve values in bytes.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-67 describes the function block for MMC_SendSdo

Figure 11-67: MMC_SendSdo function block

11.7.16.2. Function Block Code Example

```
int rc;
MMC_SENDSDO_IN         stSendSDO_in;
MMC_SENDSDO_OUT        stSendSDO_out;
//
// Inserting the structure parameters:
stSendSDO_in.lData          = 1234; //Data
stSendSDO_in.ulDataLength   = 2; //Length of the data
stSendSDO_in.usSlaveID      = 4; //The slave ID ---unused
stSendSDO_in.usIndex        = 0x6085; //COB index
stSendSDO_in.ucSubIndex     = 0;     //Defines which index value signifies the group of events
stSendSDO_in.ucService      = 1;     //Defines whether the input is a download or upload
//
rc = MMC_SendSdoCmd (hConn, iAxisRef, &stSendSDO_in, &stSendSDO_out);
if (rc != 0)
{
    HandleError();
}
```

11.8. Interpreter Command Functions

This section describes the functions (these are not function blocks) that are downloaded to the servo driver via Bin Interpreter or OS Interpreter mechanism. These functions may use RPC or IPC communication to perform the download. The purpose of these functions is to allow users of a command interpreter to access the servo driver and G-MAS via direct commands, and to move the axis via a list of commands. However, the use of the command interpreter is limited to single axis and restricts the G-MAS's capabilities.

The Get type functions perform in two ways:

- **Synchronously**
  The function does not return to the G-MAS server until a response is received from the servo driver.
- **Asynchronously**
  The function returns immediately to the G-MAS server, without waiting for a response from the servo driver. In the GMAS, when the response from the servo driver is received, it is sent via a UDP message to the library, where the connection listener thread processes the message.
  If the message is a Binary Interpreter Get command, it is processed by checking whether the Query operation mode is set to asynchronous query mode. If so, the uploaded data is kept per axis FIFO (the asynchronous query FIFO) and the axis asynchronous query FIFO index is incremented.
  However, if the Query operation mode is set to synchronous query mode, the response data is copied to the user out structure parameter.
  For every data stored in the FIFO, the FIFO index is incremented. The user can access the axis asynchronous query FIFO whenever, and retrieve data from the axis FIFO according to the axis FIFO index progress, whose value can be monitored via the *ElmoQueryOperationFIFOIndex* function.

The data returned from the Get functions, may be long or float. Therefore, storing the uploaded library data in the axes' FIFOs and enable retrieving of the data by the user is dependently performed on the specific data type saved.

Note: **For the asynchronous Get operation, a FIFO with size 1 is managed for each axis.**
If an error is sent from the servo driver because of sending an Interpreter command, the queue will be emptied and an error will be sent to the library.

11.8.1. Get Function – Asynchronous Mode

Can operate in sync and async modes. To operate in this mode, the user sets the Query operation mode to asynchronous mode,.

When a UDP message arrives to the connection UDP listener thread, the thread checks that it is a *SendRawData* – Get response, and if so, will place the data as a specific entry in a dedicated FIFO of the designated axis.

When it reaches the last entry at the FIFO, the index will not be incremented, until the user will reset the index via the ElmoQueryOperationFIFOIndexReset() function. It is responsibility of the user, to reset the Query Operation FIFO at the right time, to prevent the last record overrunning in the FIFO. The user can retrieve data from the axis Query Operation FIFO at location index in any time, via the ElmoQueryOperationFIFORetrieveData(index) function.

In order to perform Get operation in asynchronous mode, for example: et[1] for 4 axes, the following should be performed:

- ElmoGetArray(axisref1, "et", 1) to axis 1.
- ElmoGetArray(axisref2, "et", 1) to axis 2.
- ElmoGetArray(axisref3, "et", 1) to axis 3.
- ElmoGetArray(axisref4, "et", 1) to axis 4.
- ElmoQueryOperationFIFOIndex(axisref) - to return to the current index buffer location which occupies the asynchronous returned replies for a specific axis. When the current index location for one of the 1-4 axes is 1, then for that axis the asynchronous reply was received and we can start retrieving it via the, ElmoQueryOperationFIFORetrieveData(axisref, index) function.
- ElmoQueryOperationFIFORetrieveData(axisref1, 1) – Get asynchronous reply of axis 1 which located at the first entry in the axis FIFO.

The following Interpreter Command functions are described:

Interpreter Command
MMC_ElmoExecuteLabel
MMC_ElmoSetParameter
MMC_ElmoGetParameter
MMC_ElmoGetArray
MMC_ElmoGetArrayAndRetrieveData
MMC_ElmoGetParameterAndRetrieveData
MMC_ElmoSetArray
MMC_ElmoQueryOperationFIFOIndex
MMC_ElmoQueryOperationFIFORetrieveData
MMC_ElmoQueryOperationFIFOIndexReset
MMC_ElmoCall

11.8.2. MMC_ElmoExecuteLabel

Executes the user program that was downloaded via the EAS application.

```
MC_LIB_API int MMC_ElmoExecuteLabel(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_INTERPEXECUTECMD_IN* pInParam,
OUT MMC_INTERPEXECUTECMD_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_INTERPEXECUTECMD_IN** input data structure using the MMC_ElmoExecuteLabel function.

pOutParam

Points to the **MMC_INTERPEXECUTECMD_OUT** output structure receiving information, as a result of calling the MMC_ElmoExecuteLabel function.

Remarks

These functions may communicate via an RPC or IPC connection.

Scope

Note: When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).

MMC_INTERPEXECUTECMD_IN Structure

```
typedef struct{
unsigned char ucLength;
unsigned char pData[NODE_ASCII_ARRAY_MAX_LENGTH];
}MMC_INTERPEXECUTECMD_IN;
```

Parameters

ucLength

Length of the label string. Length with the precursor format of [Metronome command]##[label]. Any +ve character values.

pData

String Data with the precursor format of [Metronome command]##[label]. Any +ve character values with a maximum length of 80 bytes

[NODE_ASCII_ARRAY_MAX_LENGTH] is the node ASCII array integers with a maximum length of 80 bytes.

MMC_INTERPEXECUTECMD_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_INTERPEXECUTECMD_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-68 describes the function for MMC_ElmoExecuteLabel

Figure 11-68: MMC_ElmoExecuteLabel function

11.8.2.2. Function Code Example

```
int rc;
MMC_INTERPEXECUTECMD_IN    stInterpExCmd_in;
MMC_INTERPEXECUTECMD_OUT   stInterpExCmd_out;
//
// Inserting the structure parameters:

stInterpExCmd_in.ucLength = sizeof ("XQ##start"); //Length of the data
strcpy((char*) stInterpExCmd_in.pData,"XQ##start"); //Data

//
rc = MMC_ElmoExecuteLabel (hConn, iAxisRef, &stInterpExCmd_in, &stInterpExCmd_out);
if (rc != 0)
{
    HandleError();
}
```

11.8.3. MMC_ElmoSetParameter

Sets the Elmo drive parameter with a specific name in the servo drive.

```
MMC_LIB_API int MMC_ElmoSetParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN char cCmd[3],
IN unsigned char ucValType,
IN void* pVal
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

ucValType

Data value type, whether integer or float. Integer or float with values of 0 or 1.

pVal

Pointer to data that is to be set. Point value of a maximum of 4 bytes (Void)

Remarks

These functions may communicate via an RPC or IPC connection.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

11.8.3.1. Function Code Example

```
int rc;
//
// Inserting the structure parameters:
strcpy(cCmd,"MO");     //Name of the parameter
iArrayIdx     = 1;     //Index of array element
ucValType     = 0;     //Data value type
strcpy(pVal, "1");     //Pointer to data that is to be set
//
rc = MMC_ElmoSetParameter (hConn, iAxisRef, cCmd, iArrayIdx, ucValType, pVal);
if (rc != 0)
{
    HandleError();
}
```

11.8.4. MMC_ElmoGetParameter

Request to receive the Elmo parameters from the servo drive.

```
MMC_LIB_API int MMC_ElmoGetParameter(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN char cCmd[3],
IN unsigned char ucValType
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

ucValType

Data value type, whether integer or float. Integer or float with values of 0 or 1.

Remarks

These functions may communicate via an RPC or IPC connection.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

11.8.4.1. Function Code Example

```
int rc;
//
// Inserting the structure parameters:
strcpy(cCmd,"MO");     //Name of the parameter
ucValType     = 0;     //Data value type
//
rc = MMC_ElmoGetParameter (hConn, iAxisRef, cCmd, ucValType);
if (rc != 0)
{
    HandleError();
}
```

11.8.5. MMC_ElmoGetArray

Request to receive an element from the array parameters in the servo drive.

```
MMC_LIB_API int MMC_ElmoGetArray(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN char cCmd[3],
IN short iArrayIdx,
IN unsigned char ucValType
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iArrayIdx

Index of array element. Any +ve or –ve short integer value with a maximum of 2 bytes.

ucValType

Data value type, whether integer or float. Integer or float with values of 0 or 1.

Remarks

These functions may communicate via an RPC or IPC connection.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

11.8.5.1. Function Code Example

```
int rc;
//
// Inserting the structure parameters:
strcpy(cCmd,"MO");          //Name of the parameter
iArrayIdx     = 1;          //Index of array element
ucValType     = 0;          //Data value type
//
rc = MMC_ElmoGetArray (hConn, iAxisRef, cCmd, iArrayIdx, ucValType);
if (rc != 0)
{
    HandleError();
}
```

11.8.6. MMC_ElmoGetArrayAndRetrieveData

Synchronously requests an element from the array parameters in the servo drive and retrieves it.

```
MMC_LIB_API int MMC_ElmoGetArrayAndRetrieveData(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN char cCmd[3],
IN short iArrayIdx,
IN unsigned char ucValType,
OUT void *pVal,
OUT unsigned int* uiErrorID
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

[IN] Axis/group reference handle type returned by GetAxisRef command

cCmd[3]

[IN] Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iArrayIdx

[IN] Index of array element. Any +ve or –ve short integer value with a maximum of 2 bytes.

ucValType

[IN] Data value type, whether integer or float. Integer or float with values of 0 or 1.

pVal

[OUT] Copies a point from where to retrieve the data. Point value of a maximum of 4 bytes (Void)

uiErrorID

[OUT] Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Function init with values of 0 or +ve error_id.

Remarks

This command will necessitate waiting for the data element to be retrieved or an error returned. No other process command may be sent meanwhile.

Scope

For synchronous communication only

11.8.6.1. Function Code Example

```
int rc;
//
// Inserting the structure parameters:
strcpy(cCmd,"MO");     //Name of the parameter
iArrayIdx     = 1;     //Index of array element
ucValType     = 0;     //Data value type
//
rc = MMC_ElmoGetArrayAndRetrieveData (hConn, iAxisRef, cCmd, iArrayIdx, ucValType, pVal,
uiErrorID);
printf("Elmo Get Array and Retrieve Data Status[%ld] ErrId[%d]\n", (long int)pVal,
(int)uiErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.8.7. MMC_ElmoGetParameterAndRetrieveData

Synchronously requests a parameter in the servo drive and retrieves it.

```
MMC_LIB_API int MMC_ElmoGetParameterAndRetrieveData(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN char cCmd[3],
IN unsigned char ucValType,
OUT void *pVal,
OUT unsigned int* uiErrorID);
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

[IN] Axis/group reference handle type returned by GetAxisRef command

cCmd[3]

[IN] Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

ucValType

[IN] Data value type, whether integer or float. Integer or float with values of 0 or 1.

pVal

[OUT] Copies a point from where to retrieve the data. Point value of a maximum of 4 bytes (Void)

uiErrorID

[OUT] Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Function init with values of 0 or +ve error_id.

Remarks

This command will necessitate waiting for the data parameter to be retrieved or an error returned. No other process command may be sent meanwhile.

Scope

For synchronous communication only

11.8.7.1. Function Code Example

```
int rc;
//
// Inserting the structure parameters:
strcpy(cCmd,"MO");     //Name of the parameter
ucValType     = 0;     //Data value type
//
rc = MMC_ElmoGetParameterAndRetrieveData (hConn, iAxisRef, cCmd, ucValType, pVal, uiErrorID);
printf("Elmo Get Parameter and Retrieve Data Status[%ld] ErrId[%d]\n", (long int)pVal,
(int)uiErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.8.8. MMC_ElmoSetArray

Sets an element of the array parameters in the servo drive.

```
MMC_LIB_API int MMC_ElmoSetArray(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN char cCmd[3],
IN short iArrayIdx,
IN unsigned char ucValType,
IN void* pVal
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

[IN] Axis/group reference handle type returned by GetAxisRef command

cCmd[3]

[IN] Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iArrayIdx

[IN] Index of array element. Any +ve or –ve short integer value with a maximum of 2 bytes.

ucValType

[IN] Data value type, whether integer or float. Integer or float with values of 0 or 1.

pVal

[OUT] Pointer to data that is to be set. Point value of a maximum of 4 bytes (Void)

Remarks

This command will necessitate waiting for the data element to be retrieved or an error returned. No other process command may be sent meanwhile.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

11.8.8.1. Function Code Example

```
int rc;
//
// Inserting the structure parameters:
strcpy(cCmd,"IL");     //Name of the parameter
iArrayIdx     = 1;     //Index of array element
ucValType     = 0;     //Data value type
strcpy(pVal, "1");     //Pointer to data that is to be set
//
rc = MMC_ElmoSetArray (hConn, iAxisRef, cCmd, iArrayIdx, ucValType, pVal);
if (rc != 0)
{
    HandleError();
}
```

11.8.9. MMC_ElmoQueryOperationFIFOIndex

Returns the FIFO index.

```
MMC_LIB_API int MMC_ElmoQueryOperationFIFOIndex(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
OUT int* iReceivedMsgIdx
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

[IN] Axis/group reference handle type returned by GetAxisRef command

iReceivedMsgIdx

[OUT] Index of the number of received messages. Function init with values 0 or error_id.

Remarks

None

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

11.8.9.1. Function Code Example

```
int rc;
//
rc = MMC_ElmoQueryOperationFIFOIndex (hConn, iAxisRef, iReceivedMsgIdx);
printf("Elmo Query Operation FIFO Index Status[%ld]\n", (long int)iReceivedMsgIdx);
if (rc != 0)
{
    HandleError();
}
```

11.8.10. MMC_ElmoQueryOperationFIFORetrieveData

Request the FIFO index to retrieve data.

```
MMC_LIB_API MMC_LIB_API int MMC_ElmoQueryOperationFIFORetrieveData(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
OUT void *pVal,
OUT unsigned int* uiErrorID
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

[IN] Axis/group reference handle type returned by GetAxisRef command

pVal

[OUT] Pointer to data that is to be set. Point value of a maximum of 4 bytes (Void).

uiErrorID

[OUT] Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**. Function init with values of 0 or +ve error_id.

Remarks

If the FIFO index is 0, either an error or no data is received. If 1, data is received.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

11.8.10.1. Function Code Example

```
int rc;
//
rc = MMC_ElmoQueryOperationFIFORetrieveData (hConn, iAxisRef, pVal, uiErrorID);
printf("Elmo Query Operation FIFO Retrieve Data Status[%ld] ErrId[%d]\n", (long int)pVal,
(int)uiErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.8.11. MMC_ElmoQueryOperationFIFOIndexReset

Erases the message FIFO to 0.

```
MMC_LIB_API int MMC_ElmoQueryOperationFIFOIndexReset(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef
);
```

Motion Mode NC - Supported Distributed - Supported

Source GMAS\includes\MMC_drive_comm_API.h

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

[IN] Axis/group reference handle type returned by GetAxisRef command

Remarks

None

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

11.8.11.1. Function Code Example

```
void MMC_ElmoQueryOperationFIFOIndexReset_wrapper(int iAxisRef)
{
int rc;
//
rc = MMC_ElmoQueryOperationFIFOIndexReset (hConn, iAxisRef);
if (rc != 0)
{
    HandleError();
}
}
```

11.8.12. MMC_ElmoCall

ElmoCall is used to call a subroutine, a user program, where cCmd[3] is the name of the program

```
MMC_LIB_API int MMC_ElmoCall(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN char cCmd[3]
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_drive_comm_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

[IN] Axis/group reference handle type returned by GetAxisRef command

cCmd[3]

[IN] Name of the program limited to three characters. Any +ve character value with a maximum of 2 bytes

Remarks

Some parameters are not true variables but are direct commands to operate the servo drive. ElmoCall uses these commands to call a specific parameter from the servo drive.

Scope

All

11.8.12.1. Function Block Code Example

```
int rc;
//
// Inserting the structure parameters:
strcpy(cCmd,"BG");     //Name of the parameter
//
rc = MMC_ElmoCall (hConn, iAxisRef, cCmd);
if (rc != 0)
{
    HandleError();
}
```

11.9. EtherNetIP Communication

This section describes the EtherNet/IP (Ethernet Industrial Protocol), which is a communications protocol developed by Rockwell Automation, managed by ODVA and designed for use in process control and other industrial automation applications e.g. water processing plants, manufacturing facilities, and other utilities. It canbe used in automation networks, which tolerate some amount of non-determinism. This is because the Ethernet physical media, by definition, is not deterministic. EtherNet/IP is most commonly used in the US and Asia for communication to and from Rockwell Automation's Allen-Bradley-control systems.

ODVA manages the protocol and assures multi-vendor system interoperability. The transfer of basic I/O data is via the UDP-based implicit messaging (port 2222). Uploading and downloading of parameters via TCP (i.e. explicit messaging port 44818). The EtherNet/IP application layer protocol is based on the Common Industrial Protocol (CIP) standard used in DeviceNet. Elmo uses the PyramidSolutions EIP adapter stack (EADK) used in the Gold Maestro. It was compiled for Elmo's Linux toolchain.

11.9.1. Terminology

The devices are:

Devices	Explanation
Master (Scanner)	Initiates communications with adapter devices. A scanner is typically the most complex type of EtherNet/IP device, as it must manage issues such as configuration, connections, of the adapter device. A Rockwell PLC is an example of a scanner.
Slave (Adapter)	Receives communication connection requests from a scanner and then produces its I/O data at the requested rate. An adapter can be a simple digital input device or more complex such as a modular pneumatic valve system, the G-MAS.

The message types involved in EtherNetIP are:

Messages	Explanation
Explicit messaging (information)	Non time-critical data transfers that are a relatively large packet size for short-lived connections between originator and single target device. The data is transferred via TCP/IP.
Implicit messaging (I/O data)	Time-critical data transfers usually consist of small packets. I/O data exchanges are long term lived connections between single originator to one or more target devices. I/O data packets are UDP packets.

11.9.1.1. Assemblies

An Ethernet /IP object is used for I/O message communication. There are three types of Assemblies, which use implicit messaging:

Assembly	Explanation
Production	produce (sends) data
Consumption	consumes (receives) data
Configuration	general configuration

11.9.1.2. Tags

A Tag defines the name for a single or array of variable(s). there are two types of Tags:

TAGs	Explanation
Adapter	Data that resides in the G-MAS and is available for other devices on the Ethernet/IP.
Device	Data that resides on other devices on the Ethernet/IP, e.g. PLC.

11.9.1.3. Data Types

The Following data Types are supported (Assemblies and Tags):

Data Type	Explanation
BOOL(Tag only)	unsigned char (1 byte)
SINT	signed char (1 byte)
INT	short int (2 bytes)
DINT	int (4 bytes)
REAL	float (4 bytes)

11.9.2. Configuring the Ethernet IP Device as Adapter

Identical Configurations must reside on the Master and Slave (G-MAS) side. The assembly parameters should be of one to three directional types: Consumer, Producer or Configuration, operating in their specific directions. The opposing direction must reside on the Master side. The assemblies are identified by the InstanceID, and have member data. The member data is exchanged every I/O data exchange (initiated by scanner). The Assembly definition is created in the Properties window as shown in the Maestro assembly configuration window.

Tags (Slave (Adapter) or Device) may be defined as arrays or single dimension, and are sent as soon as they are modified. Tags have names, which are defined in the XML Configuration File.

This XML configuration file contains all of the Ethernet/IP data, e.g. Paths, Tags and Assembly data, and is similar to the structure of the Maestro XML File.

```
  <?xml version="1.0" ?>
- <EIPADAPTER>
  - <EIPDEVICES>
    - <EIPDEVICE NAME="ggg" NETWORKPATH="192.168.1.1">
        <EIPTAG TYPE="REAL" NAME="jjjjjj" SIZE="4,6" />
      </EIPDEVICE>
    </EIPDEVICES>
  - <EIPASSEMBLIES>
    - <EIPASSEMBLY NAME="outAss" INSTANCE="1" SIZE="4" OFFSET="4294967295" COMFORMAT="DINT" TYPE="8">
        <EIPASSEMBLY_MEMBER ID="0" SIZE="32" OFFSET="0" COMFORMAT="DINT" NAME="Param00" />
        <EIPASSEMBLY_MEMBER ID="1" SIZE="32" OFFSET="32" COMFORMAT="DINT" NAME="Param01" />
        <EIPASSEMBLY_MEMBER ID="2" SIZE="32" OFFSET="64" COMFORMAT="DINT" NAME="Param02" />
        <EIPASSEMBLY_MEMBER ID="3" SIZE="32" OFFSET="96" COMFORMAT="DINT" NAME="Param03" />
      </EIPASSEMBLY>
    - <EIPASSEMBLY NAME="inAss" INSTANCE="2" SIZE="4" OFFSET="4294967295" COMFORMAT="DINT" TYPE="4">
        <EIPASSEMBLY_MEMBER ID="0" SIZE="32" OFFSET="0" COMFORMAT="DINT" NAME="Param00" />
        <EIPASSEMBLY_MEMBER ID="0" SIZE="32" OFFSET="0" COMFORMAT="DINT" NAME="Param00" />
        <EIPASSEMBLY_MEMBER ID="0" SIZE="32" OFFSET="0" COMFORMAT="DINT" NAME="Param00" />
        <EIPASSEMBLY_MEMBER ID="0" SIZE="32" OFFSET="0" COMFORMAT="DINT" NAME="Param00" />
      </EIPASSEMBLY>
    </EIPASSEMBLIES>
  - <EIPADAPTERTAGS>
      <EIPADAPTERTAG TYPE="INT" NAME="KUKU111" SIZE="2,3" />
      <EIPADAPTERTAG TYPE="BOOL" NAME="dsfsdf" SIZE="1" />
    </EIPADAPTERTAGS>
  </EIPADAPTER>
```

Figure 11-69 describes the G-MAS operation block diagram using EtherNetIP.

Figure 11-69: G-MAS Operation block diagram

The EADK is part of the EIP_LIB (G-MAS Library), with the following advantages:

- Less Overhead when reading/writing data. Faster.
- Less events, context switching, etc …
- Easy debugging.

As such, the Assembly Tags reside both in the G-MAS and other device memory according to the following:

Tags	Explanation
Device	Reside at the other device memory
Adapter	Reside at the G-MAS memory

When Initialized, the MMC_EipInit performs the following:

1. Calls relevant EIP stack APIs that:
   a. Initialize callbacks
   b. Creates listening thread
2. Initializes assemblies and Tags by parsing XML.
3. Sets assemblies / member structures in the stack.

11.9.2.1. Assembly Setting Data

The user calls MMC_EipSetAssembly() – with input of Producing assembly instance, and then sends the data with the InstanceID with assembly data structure. If an error occurs – an event is sent. The message is sent at the next scanner cycle.

Then user then calls MMC_EipGetAssembly() – with input of Consuming assembly instance. The function returns the data to a local data structure.

11.9.2.2. New Assembly Received Event

The EADK invokes a callback when new data (modified) is received. This callback calls the same default callback as in the User program. User program can then read the assembly data (Instead of polling).

The device Tags are set by calling the EipSetAsyncTag API. If no immediate error occurs, then the message is sent. If an error occurs while sending the message – an event is invoked and the user is notified.

11.9.2.2.a Modes

Mode	Explanation
Async.	Does not wait for the EipReadTag. The EipGetAsyncTag function reads the EipReadTag, but not in the immediate sequence
Sync.	Waits for the EipReadTag and employs the EipGetAsyncTag function to read the EipReadTag in phases 1 to 3 in sequence

The User calls the EipGetAsyncTag API. If no immediate error occurs, then the message is sent and a ReqID is returned (Max 100 concurrent messages). Once the stack replies, a flag is set that the message arrived. User may poll or receive an event that a reply has arrived, but is not pending.

User calls the EIPReadTag(ReqID) to retrieve the reply. If an error occurs while sending the message, an event is invoked and the user is notified. This is similar to the Async mode, except that the API function waits until the requested tag data arrives.

Once a tag arrives and it is pending an event, an event is set to the calling function. Similar to the ElmoGetSyncParameter.

11.9.2.3. Timeouts and errors

Timeouts and errors are also handled using Adapter Tags. These are tags that reside in the G-MAS memory space. A device writes/reads to the adapter tags at the stack level and does not need User Program Intervention. These tags are built at EIP initialization, and their responses are handled in the EADK callback function.

1. The User calls the clientRegObjectsForClientProc API and allocates memory accordingly.
2. Stack sends NM_CLIENT_OBJECT_REQUEST_RECEIVED events. Once the event is received by the stack, service type is requested by clientGetClientRequest API
3. Stack sends requested service (read/write) with ID. Service is completed and allocated data sent to stack.
4. User can update/read allocated data asynchronically. If an error occurs while sending the message – an event is invoked and the user is notified.

11.9.3. Ethernet/IP Setup

GMAS operation as an Ethernet/IP adapter (Slave) is actually an application, which runs on the G-MAS. As an initial stage it loads and parses an XML file, which defines a specific configuration on the Scanner (e.g. Rockwell PLC). The name of the file can be any name as long as the extension is xml, the scheme is correct (see XML Setup File Sample below), and the adapter program is aware of that name and the file full path.

```
<?xml version="1.0" encoding="utf-8"?>
<EIPCONFIG>
<EIPDEVICES DEVNUM="2">
        <EIPDEVICE NAME="dev_plc2gmas" NETWORKPATH="10.10.20.229,1,0" TYPE="DINT" SIZE="1"/>
        <EIPDEVICE NAME="dev_gmas2plc" NETWORKPATH="10.10.20.229,1,0" TYPE="DINT" SIZE="1"/>
    </EIPDEVICES>
    <EIPASSEMBLIES NUM="3">
        <EIPASSEMBLY NAME="asm_gmas2plc" INSTANCE="1" SIZE="1" OFFSET="0" COMFORMAT="DINT" TYPE="9"/>
        <EIPASSEMBLY NAME="asm_plc2gmas" INSTANCE="2" SIZE="1" OFFSET="4" COMFORMAT="DINT" TYPE="5"/>
        <EIPASSEMBLY NAME="cfgAss" INSTANCE="3" SIZE="1" OFFSET="8" COMFORMAT="DINT" TYPE="17"/>
    </EIPASSEMBLIES>
    <EIPADAPTERS NUM="2">
        <EIPADAPTER TYPE="DINT" NAME="adp_plc2gmas" SIZE="1"/>
        <EIPADAPTER TYPE="DINT" NAME="adp_gmas2plc" SIZE="1"/>
    </EIPADAPTERS>
</EIPCONFIG>
```

11.9.3.1. XML Structure

The XML file consist of three blocks for devices (EIPDEVICES), assemblies (EIPASSEMBLIES) and adapters (EIPADAPTERS). Each of them has an attribute NUM, which declares the amount of elements it contains.

EIPDEVICE Attributes

NAME

Name of variable bound to PLC tag. The EIP program on the G-MAS retrieves a tag reference by this name.

NETWORKPATH

<IP address>,1,0. The 1,0 is specifically required by the Rockwell PLC scanner. It may be redundant for other scanners.

TYPE

Variable type: SINT (char, 1 byte), INT (short, 2 bytes), DINT (int, 4 bytes), REAL (float, 4 bytes)

SIZE

Variable dimension. One may declare a device tag as an array.

EIPASSEMBLY Attributes

NAME

Name of variable bound to PLC assembly (IO). The EIP program on the G-MAS retrieves a tag reference by this name.

INSTANCE

Unlike other tags (adapter, device), this tag reference may also be retrieved by instance (optionally).

SIZE

Variable dimension. One may declare an assembly tag as an array.

OFFSET

Unlike other tags (adapter, device) for all assemblies which are mapped into one flat buffer in memory, the user should map each **assembly** variable into this buffer according to its type (see COMFORMAT) and dimension. The two variables of type int (DINT) should be mapped. For example, if the first offset is 0 then the second will be 4, since DINT is four byte length.

COMFORMAT

Variable type format: SINT (char, 1 byte), INT (short, 2 bytes), DINT (int, 4 bytes), REAL (float, 4 bytes).

TYPE

This attribute sets the EIP library mode of operation and it should be constant for this stage:

9 - Assembly that produces data on the network (GMAS 2 PLC)

5 - Assembly that consumes data from the network (PLC 2 GMAS)

17 - Assembly that will be used to store configuration.

EIPADAPTER Attributes

NAME

Name of variable bound to the PLC tag. The EIP program on the G-MAS retrieves the tag reference by this name.

TYPE

Variable type: SINT (char, 1 byte), INT (short, 2 bytes), DINT (int, 4 bytes), REAL (float, 4 bytes)

SIZE

Variable dimension. One may declare an adapter tag as an array.

11.10. EtherNetIP Functions

The following EtherNetIP communication functions are described:

EtherNetIP Communication
EipGetAdpTagRefByName
EipWriteAdpTag
EipReadAdpTag
EipGetAssemblyRefByInstance
EipGetAssemblyRefByName
EipSetAssembly
EipGetAssembly
EipGetDevTagRefByName
EipSetDevTag
EipGetDevTag
EipReadDevTagData
EipSyncGetDevTag
EipCheckDevTagReply
EipOpenSession
EIPCloseSession
EipCreate
EipDestroy

11.10.1. EipGetAdpTagRefByName

Returns adapter tag reference index according to its name.

```
int EipGetAdpTagRefByName(
IN EIP_REFBYNAME_IN *pInParam,
OUT EIP_REFBYNAME_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_REFBYNAME_IN** input data structure using the EipGetAdpTagRefByName function. References the adapter tag name as declared in the XML configuration file.

pOutParam

Points to the **EIP_REFBYNAME_OUT** output structure receiving information as a result of calling the EipGetAdpTagRefByName function. Index reference to the adapter tag.

Remarks

None

Scope

All

EIP_REFBYNAME_IN Structure

```
typedef struct {
char cName[NAME_MAX_LENGTH];
}EIP_REFBYNAME_IN;
```

Parameters

cName

Tag/assembly name as declared in XML configuration file. The variable [NAME_MAX_LENGTH] is limited to 80 characters.

EIP_REFBYNAME_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned short usTagRef;
}EIP_REFBYNAME_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

Figure 11-70 describes the function block for EipGetAdpTagRefByName

Figure 11-70: EipGetAdpTagRefByName function

11.10.2. EipWriteAdpTag

Returns adapter tag reference index according to its name.

```
int EipWriteAdpTag(
IN EIP_WRITEDATA_IN *pInParam,
OUT EIP_WRITEDATA_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_WRITEDATA_IN** input data structure using the EipWriteAdpTag function. Reference index to the adapter tag.

pOutParam

Points to the **EIP_WRITEDATA_OUT** output structure receiving information as a result of calling the EipWriteAdpTag function. Data to be written to the referenced adapter tag.

Remarks

None

Scope

All

EIP_WRITEDATA_IN Structure

```
typedef struct{
unsigned short usTagRef;
char buffer[MAX_REQUEST_DATA_SIZE];
}EIP_WRITEDATA_IN;
```

Parameters

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

buffer

Buffer that holds returned data. The variable [MAX_REQUEST_DATA_SIZE] is limited to 504 characters.

EIP_WRITEDATA_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
int iReqid;
}EIP_WRITEDATA_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

iReqid

Adapter tag request id

Figure 11-71 describes the function block for EipWriteAdpTag

Figure 11-71: EipWriteAdpTag function

11.10.3. EipReadAdpTag

Copies adapter tag data from memory into input buffer.

```
int EipReadAdpTag(
IN EIP_READADPTAG_IN *pInParam,
OUT EIP_READADPTAG_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_READADPTAG_IN** input data structure using the EipReadAdpTag function. Reference index to adapter tag.

pOutParam

Points to the **EIP_READADPTAG_OUT** output structure receiving information as a result of calling the EipReadAdpTag function. Buffer to hold adapter data.

Remarks

None

Scope

All

EIP_READADPTAG_IN Structure

```
typedef struct{
unsigned short usTagRef;
}EIP_READADPTAG_IN;
```

Parameters

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

EIP_READADPTAG_OUT Structure

```
typedef struct {
unsigned short usStatus;
short usErrorID;
char buffer[MAX_REQUEST_DATA_SIZE];
}EIP_READADPTAG_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

buffer

Buffer that holds returned data. The variable [MAX_REQUEST_DATA_SIZE] is limited to 504 characters.

Figure 11-72 describes the function block for EipReadAdpTag

Figure 11-72: EipReadAdpTag function

11.10.4. EipGetAssemblyRefByInstance

Finds the asm_instance and applies a reference to this instance.

```
int EipGetAssemblyRefByInstance(
IN EIP_REFBYINSTANCE_IN *pInParam,
OUT EIP_REFBYINSTANCE_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_REFBYINSTANCE_IN** input data structure using the EipGetAssemblyRefByInstance function, where asm_instance is an existing assembly instance.

pOutParam

Points to the **EIP_REFBYINSTANCE_OUT** output structure receiving information as a result of calling the EipGetAssemblyRefByInstance function, where the referenced index relates to the requested assembly.

Remarks

None

Scope

All

EIP_REFBYINSTANCE_IN Structure

```
typedef struct{
int iInstance;
}EIP_REFBYINSTANCE_IN;
```

Parameters

iInstance

Assembly instance as declared in the XML configuration file.

EIP_REFBYINSTANCE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned short usTagRef;
}EIP_REFBYINSTANCE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

Figure 11-73 describes the function block for EipGetAssemblyRefByInstance

Figure 11-73: EipGetAssemblyRefByInstance function

11.10.5. EipGetAssemblyRefByName

This function returns the assembly reference index according to its name.

```
int EipGetAssemblyRefByName(
IN EIP_REFBYNAME_IN *pInParam,
OUT EIP_REFBYNAME_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_REFBYNAME_IN** input data structure using the EipGetAssemblyRefByName function, where asm_name is the assembly name as declared in the XML configuration file.

pOutParam

Points to the **EIP_REFBYNAME_OUT** output structure receiving information as a result of calling the EipGetAssemblyRefByName function, where the referenced index relates to the adapter tag.

Remarks

None

Scope

All

EIP_REFBYNAME_IN Structure

```
typedef struct{
char cName[NAME_MAX_LENGTH];
}EIP_REFBYNAME_IN;
```

Parameters

cName

Tag/assembly name as declared in the XML configuration file. The variable [NAME_MAX_LENGTH] is limited to 80 characters.

EIP_REFBYNAME_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned short usTagRef;
}EIP_REFBYNAME_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

Figure 11-74 describes the function block for EipGetAssemblyRefByName

Figure 11-74: EipGetAssemblyRefByName function

11.10.6. EipSetAssembly

Fills the assembly data with out_buff data and sends it through EthernetIP.

```
int EipSetAssembly(
IN EIP_SETASSEMBLY_IN *pInParam,
OUT EIP_SETASSEMBLY_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_SETASSEMBLY_IN** input data structure using the E EipSetAssembly function, where the input is an instance of the requested assembly.

pOutParam

Points to the **EIP_SETASSEMBLY_OUT** output structure receiving information as a result of calling the EipSetAssembly function, where the output relates to the buffer which holds data sent to the assembly.

Remarks

None

Scope

All

EIP_SETASSEMBLY_IN Structure

```
typedef struct{
unsigned short usInstance;
char buffer[MAX_REQUEST_DATA_SIZE];
}EIP_SETASSEMBLY_IN;
```

Parameters

buffer

Buffer that holds returned data. The variable [MAX_REQUEST_DATA_SIZE] is limited to 504 characters.

EIP_SETASSEMBLY_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
int iReqid;
}EIP_SETASSEMBLY_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

iReqid

Assembly request ID. Integer value.

Figure 11-75 describes the function block for EipSetAssembly

Figure 11-75: EipSetAssembly function

11.10.7. EipGetAssembly

Copies an assembly data identified by instance to in_buff.

```
int EipGetAssembly(
IN EIP_GETASSEMBLY_IN *pInParam,
OUT EIP_GETASSEMBLY_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_GETASSEMBLY_IN** input data structure using the EipGetAssembly function identified by the instance of requested assembly.

pOutParam

Points to the **EIP_GETASSEMBLY_OUT** output structure receiving information as a result of calling the EipGetAssembly function using the buffer to hold incoming assembly data.

Remarks

None

Scope

All

EIP_GETASSEMBLY_IN Structure

```
typedef struct{
unsigned short usInstance;
}EIP_GETASSEMBLY_IN;
```

Parameters

usInstance

Assembly instance reference. Any +ve value accepted.

EIP_GETASSEMBLY_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
char buffer[MAX_REQUEST_DATA_SIZE];
}EIP_GETASSEMBLY_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

buffer

Buffer that holds returned data. The variable [MAX_REQUEST_DATA_SIZE] is limited to 504 characters.

Figure 11-76 describes the function block for EipGetAssembly

Figure 11-76: EipGetAssembly function

11.10.8. EipGetDevTagRefByName

This function returns device tag reference index according to its name.

```
int EipGetDevTagRefByName(
IN EIP_REFBYNAME_IN *pInParam,
OUT EIP_REFBYNAME_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_REFBYNAME_IN** input data structure using the EipGetDevTagRefByName function. References the device tag name as declared in XML configuration file.

pOutParam

Points to the **EIP_REFBYNAME_OUT** output structure receiving information as a result of calling the EipGetDevTagRefByName function, and the index reference to the device tag.

Remarks

None

Scope

All

EIP_REFBYNAME_IN Structure

```
typedef struct {
char cName[NAME_MAX_LENGTH];
}EIP_REFBYNAME_IN;
```

Parameters

cName

Tag/assembly name as declared in XML configuration file. The variable [NAME_MAX_LENGTH] is limited to 80 characters.

EIP_REFBYNAME_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
unsigned short usTagRef;
}EIP_REFBYNAME_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

Figure 11-77 describes the function block for EipGetAdpTagRefByName

Figure 11-77: EipGetAdpTagRefByName function

11.10.9. EipSetDevTag

Updates device tag data and sends it to the EIP device.

```
int EipSetDevTag(
IN EIP_SETDEVTAG_IN *pInParam,
OUT EIP_SETDEVTAG_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_SETDEVTAG_IN** input data structure using the EipSetDevTag function that is index referenced to the device tag.

pOutParam

Points to the **EIP_SETDEVTAG_OUT** output structure receiving information as a result of calling the EipSetDevTag function using the buffer which holds data to be sent to the device tag.

Remarks

None

Scope

All

EIP_SETDEVTAG_IN Structure

```
typedef struct{
unsigned short usTagRef;
char buffer[MAX_REQUEST_DATA_SIZE];
}EIP_SETDEVTAG_IN;
```

Parameters

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

buffer

Buffer that holds returned data. The variable [MAX_REQUEST_DATA_SIZE] is limited to 504 characters.

EIP_SETDEVTAG_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
int iReqid;
}EIP_SETDEVTAG_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

iReqid

Device tag request id. Integer value.

Figure 11-78 describes the function block for EipSetDevTag

Figure 11-78: EipSetDevTag function

11.10.10. EipGetDevTag

Sends request to the EIP device to read specific device tag.

```
int EipGetDevTag(
IN EIP_GETDEVTAG_IN *pInParam,
OUT EIP_GETDEVTAG_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_GETDEVTAG_IN** input data structure using the EipGetDevTag function that is index referenced to the device tag.

pOutParam

Points to the **EIP_GETDEVTAG_OUT** output structure receiving information as a result of calling the EipGetDevTag function using the requested ID to be used to read data when received.

Remarks

None

Scope

All

EIP_GETDEVTAG_IN Structure

```
typedef struct{
unsigned short usTagRef;
}EIP_GETDEVTAG_IN;
```

Parameters

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

EIP_GETDEVTAG_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
int iReqid;
}EIP_GETDEVTAG_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

iReqid

Device tag request ID. Integer value.

Figure 11-79 describes the function block for EipGetDevTag

Figure 11-79: EipGetDevTag function

11.10.11. EipReadDevTagData

Reads and stores device tag data received from an EIP device, as a response to user request.

```
int EipReadDevTagData(
IN EIP_READDEVTAG_IN *pInParam,
OUT EIP_READDEVTAG_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_READDEVTAG_IN** input data structure using the EipReadDevTagData function that is index referenced to the device tag.

pOutParam

Points to the **EIP_READDEVTAG_OUT** output structure receiving information as a result of calling the EipReadDevTagData function using the buffer to hold the incoming device tag data.

Remarks

None

Scope

All

EIP_READDEVTAG_IN Structure

```
typedef struct{
unsigned short usTagRef;
}EIP_READDEVTAG_IN;
```

Parameters

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

EIP_READDEVTAG_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
char buffer[MAX_REQUEST_DATA_SIZE];
}EIP_READDEVTAG_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

buffer

Buffer that holds returned data. The variable [MAX_REQUEST_DATA_SIZE] is limited to 504 characters.

Figure 11-80 describes the function block for EipReadDevTagData

Figure 11-80: EipReadDevTagData function

11.10.12. EipSyncGetDevTag

Sends a request to read device tag data, and waits for a response to be received.

```
int EipSyncGetDevTag(
IN EIP_GETSYNC_IN *pInParam,
OUT EIP_GETSYNC_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_GETSYNC_IN** input data structure using the EipSyncGetDevTag function that is index referenced to the device tag.

pOutParam

Points to the **EIP_GETSYNC_OUT** output structure receiving information as a result of calling the EipSyncGetDevTag function using the buffer to hold the incoming device tag data.

Remarks

None

Scope

All

EIP_GETSYNC_IN Structure

```
typedef struct{
unsigned short usTagRef;
}EIP_GETSYNC_IN;
```

Parameters

usTagRef

Tag/Assembly index reference. Any +ve value accepted.

EIP_GETSYNC_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
char buffer[MAX_REQUEST_DATA_SIZE];
}EIP_GETSYNC_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

buffer

Buffer that holds returned data. The variable [MAX_REQUEST_DATA_SIZE] is limited to 504 characters.

Figure 11-81 describes the function block for EipSyncGetDevTag.

Figure 11-81: EipSyncGetDevTag function

11.10.13. EipCheckDevTagReply

Check that a reply has been received for a specific device tag request.

```
void EipCheckDevTagReply(
IN EIP_CHECKREPLY_IN *pInParam,
OUT EIP_CHECKREPLY_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_CHECKREPLY_IN** input data structure using the EipCheckDevTagReply function that is the device tag request ID.

pOutParam

Points to the **EIP_CHECKREPLY_OUT** output structure receiving information as a result of calling the EipCheckDevTagReply function.

Remarks

None

Scope

All

EIP_CHECKREPLY_IN Structure

```
typedef struct{
int iReqid;
}EIP_CHECKREPLY_IN;
```

Parameters

iReqid

Device tag request ID. Integer value.

EIP_CHECKREPLY_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
short sReplyStatus;
}EIP_CHECKREPLY_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

sReplyStatus

Returned reply status flag

Figure 11-82 describes the function block for EipCheckDevTagReply.

Figure 11-82: EipCheckDevTagReply function

11.10.14. EipOpenSession

Initialize and start an EIP session in order to be able to use EthernetIP.

```
int EipOpenSession(
IN EIP_CALLBACK_FUNC pCallBackFunc,
IN EIP_OPEN_SESSION_IN *pInParam,
OUT EIP_OPEN_SESSION_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pCallBackFunc

The callback function is returned from the stack of the Ethernet IP library originating from the G-MAS, together with an EIP type event. The Buffer index may be 0, but the EIP event data is positioned at index 20.

pInParam

Points to the **EIP_OPEN_SESSION_IN** input data structure using the EipOpenSession function.

pOutParam

Points to the **EIP_OPEN_SESSION_OUT** output structure receiving information as a result of calling the EipOpenSession function.

Remarks

None

Scope

All

EIP_OPEN_SESSION_IN Structure

```
typedef struct{
char cDummy;
}EIP_OPEN_SESSION_IN;
```

Parameters

cDummy

Dummy value

EIP_OPEN_SESSION_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}EIP_OPEN_SESSION_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

Figure 11-83 describes the function block for EipOpenSession.

Figure 11-83: EipOpenSession function

11.10.15. EIPCloseSession

Close an EtherNETIP session and free allocated memory before terminating program.

```
int EIPCloseSession(
IN EIP_CLOSE_SESSION_IN *pInParam,
OUT EIP_CLOSE_SESSION_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_CLOSE_SESSION_IN** input data structure using the EIPCloseSession function.

pOutParam

Points to the **EIP_CLOSE_SESSION_OUT** output structure receiving information as a result of calling the EIPCloseSession function.

Remarks

None

Scope

All

EIP_CLOSE_SESSION_IN Structure

```
typedef struct{
char cDummy;
}EIP_OPEN_SESSION_IN;
```

Parameters

cDummy

Dummy value

EIP_CLOSE_SESSION_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}EIP_OPEN_SESSION_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

Figure 11-84 describes the function block for EIPCloseSession.

Figure 11-84: EIPCloseSession function

11.10.16. EipCreate

Create an EtherNETIP session.

```
int EipCreate(
IN EIP_CREATE_IN *pInParam,
OUT EIP_CREATE_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_CREATE_IN** input data structure using the EipCreate function.

pOutParam

Points to the **EIP_CREATE_OUT** output structure receiving information as a result of calling the EipCreate function.

Remarks

None

Scope

All

EIP_CREATE_IN Structure

```
typedef struct{
char cPath[80];
}EIP_CREATE_IN;
```

Parameters

cPath[80]

Path of the EtherNETIP session. Value to maximum of 80 characters.

EIP_CREATE_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}EIP_CREATE_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

Figure 11-85 describes the function block for EipCreate.

Figure 11-85: EipCreate function

11.10.17. EipDestroy

Kills the EtherNETIP session.

```
int EipDestroy(
IN EIP_DESTROY_IN *pInParam,
OUT EIP_DESTROY_OUT *pOutParam
);
```

Motion Mode	NC – Not relevant	Distributed – not relevant
Source	GMAS\includes\MMC_EIP_API.h	

Function Parameters

pInParam

Points to the **EIP_DESTROY_IN** input data structure using the EipDestroy function.

pOutParam

Points to the **EIP_DESTROY_OUT** output structure receiving information as a result of calling the EipDestroy function.

Remarks

None

Scope

All

EIP_DESTROY_IN Structure

```
typedef struct{
char dummy;
}EIP_DESTROY_IN;
```

Parameters

dummy

Dummy value

EIP_DESTROY_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}EIP_DESTROY_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in section **6.10 EtherNetIP Communication Error Codes on page 6-465**. Displays an error code as -ve integers.

Figure 11-86 describes the function block for EipDestroy.

Figure 11-86: EipDestroy function

11.10.18. Functions and Implementation Example

```
/*
* EIPSample.c
*/

#include "EIPSample.h"
#include <expat.h>
#include "EIP_API.h"
#include "datastructs.h"

/* ////////////////////////////////////////////////////////////////////// */
/*                              Globals                                 */
/* ////////////////////////////////////////////////////////////////////// */
float new_pos,position, velocity;
int axis_ref;
int  pos_cmd_id ,servo_cmd_id;
char new_servo_cmd,servo_cmd;

unsigned long  ulState ;
unsigned int err_stat=0;
int val;

/* ////////////////////////////////////////////////////////////////////// */
/*                        external     Globals                          */
/* ////////////////////////////////////////////////////////////////////// */
extern char g_cfg_name[256];

/* ////////////////////////////////////////////////////////////////////// */
/*                              Function Prototypes                     */
/* ////////////////////////////////////////////////////////////////////// */
int EIPSample();
int EipReadInputs();
void EipInit();
int EipWriteStatus();

/* ////////////////////////////////////////////////////////////////////// */
/*                              External Functions                      */
/* ////////////////////////////////////////////////////////////////////// */
extern int MyCallbackFunc (unsigned char*, short,void*) ;
extern int PowerCmdOn();
extern int PowerCmdOff();
extern int MoveAbsolute(int axis_idx,double position , float velocity);


/* ////////////////////////////////////////////////////////////////////// */
/*                              Function Implementation                 */
/* ////////////////////////////////////////////////////////////////////// */

/*************************************************************************
* FUNCTION:             EIPSample()
* DESCRIPTION:
* INPUTS:               N/A
* OUTPUTS:
* RETURN VALUE:
*************************************************************************/
int EIPSample()
{
    int indx;

    axis_ref=0;
    sr_data_ref=0;
    pos_cmd_id =0;
    servo_cmd_id =0;
    velocity = 2000;

    EipInit();
```

```
    indx=0;
    while(1)
    {
    indx++;

    EipReadInputs();

        if (!(ulState & NC_AXIS_VALID_MASK))
        {
            usleep(10000);
            continue;
        }

    if (new_servo_cmd != servo_cmd)
    {
        if ((ulState & NC_AXIS_STAND_STILL_MASK) && (!new_servo_cmd))
        {
            printf("Power OFF\n");
            PowerCmdOff();
        }

        if ((ulState & NC_AXIS_DISABLED_MASK) && (new_servo_cmd))
        {
            printf("Power ON\n");
            PowerCmdOn();
        }
    }

    if (new_pos !=position)
    {
        if (ulState & NC_AXIS_STAND_STILL_MASK)
        {
            printf("Move Absolute from %f to %f\n",position,new_pos);
            MoveAbsolute(axis_ref, new_pos , velocity);
            position = new_pos;
        }
    }

    EipWriteStatus();

        if(gTerminateFlag)
        {
            break;
        }
    }

    return 0;
}

/*****************************************************************************
* FUNCTION:             EipReadInputs()
* DESCRIPTION:
* INPUTS:               N/A
* OUTPUTS:
* RETURN VALUE:
*****************************************************************************/
int EipReadInputs()
{

    static int read_position =1;
    static int read_servo =1;

    /* ---------------------------- */
    /* Asynchronous position request */
    /* ---------------------------- */
    if (read_position)
    {
    get_devtag_in.usTagRef = pos_cmd_ref;
```

```
        EipGetDevTag(&get_devtag_in, &get_devtag_out);

        if(get_devtag_out.usErrorID !=0)
    {
            printf("position request error\n");
            gTerminateFlag =1;
            return -1;
        }

        pos_cmd_id = get_devtag_out.iReqid;
        read_position =0;
    }

    /* ------------------------------- */
    /* Asynchronous stop command check */
    /* ------------------------------- */
    if (read_servo)
    {
    get_devtag_in.usTagRef = servo_cmd_ref;
    EipGetDevTag(&get_devtag_in, &get_devtag_out);

        if(get_devtag_out.usErrorID !=0)
    {
            printf("servo request error\n");
            gTerminateFlag =1;
        return -1;
        }
        servo_cmd_id = get_devtag_out.iReqid;
        read_servo=0;
    }

    if (MMC_ElmoGetParameterAndRetrieveData(g_conn_hndl,axis_ref,"SR",0,&val,&err_stat)!=0)
    {
        printf("error reading SR paramter\n");
        return -1;
    }

    /* ----------------------------- */
    /* position request reply  check */
    /* ----------------------------- */
    check_reply_in.iReqid = pos_cmd_id;
    EipCheckDevTagReply(&check_reply_in,&check_reply_out);

   if (check_reply_out.sReplyStatus)
    {

    read_dev_in.usTagRef = pos_cmd_ref;
    EipReadDevTagData(&read_dev_in,&read_dev_out);

    if(read_dev_out.usErrorID!=0)
    {
            printf("reading position data failure %d\n",read_dev_out.usErrorID);

            gTerminateFlag =1;
            return -1;
        }

    new_pos = *(float *)(read_dev_out.buffer);
    read_position =1;
    }

    /* ------------------------------ */
    /* stop flag request reply  check */
    /* ------------------------------ */
   check_reply_in.iReqid = servo_cmd_id;
    EipCheckDevTagReply(&check_reply_in,&check_reply_out);

    if (check_reply_out.sReplyStatus)
```

```
    {
        read_dev_in.usTagRef = servo_cmd_ref;
    EipReadDevTagData(&read_dev_in,&read_dev_out);

    if(read_dev_out.usErrorID!=0)
        {
            printf("reading stop flag failure %d\n",read_dev_out.usErrorID);
            gTerminateFlag =1;
            return -1;
        }

        new_servo_cmd = *read_dev_out.buffer;
        read_servo =1;
    }


    if (MMC_ReadStatusCmd(g_conn_hndl, axis_ref, &read_status_in,&read_status_out) != 0)
    {
        printf("%s Read Status Failed\n",__func__);
        return -1;
    }
    ulState = read_status_out.ulState ;

    return 0;

}

/***************************************************************************
* FUNCTION:             EipReadInputs()
* DESCRIPTION:
* INPUTS:               N/A
* OUTPUTS:
* RETURN VALUE:
***************************************************************************/
int EipWriteStatus()
{
    MMC_ReadActualPositionCmd(g_conn_hndl,  axis_ref, &read_pos_in,    &read_pos_out);
    MMC_ReadActualVelocityCmd(g_conn_hndl,  axis_ref, &read_vel_in, &read_vel_out);

    write_adp_in.usTagRef =axis_data_ref ;
    *(float*)&write_adp_in.buffer[0] = read_pos_out.dbPosition;
    EipWriteAdpTag (&write_adp_in,&write_adp_out);


    write_adp_in.usTagRef =sr_data_ref ;
    *(int*)&write_adp_in.buffer[0] = val;
    EipWriteAdpTag (&write_adp_in,&write_adp_out);

    //position=new_pos;
    servo_cmd = new_servo_cmd;

    set_asm_in.usInstance =1;
    *(float*)&set_asm_in.buffer = read_vel_out.dVelocity;
    EipSetAssembly(&set_asm_in, &set_asm_out);

    return 0;
}

/***************************************************************************
* FUNCTION:             EipReadInputs()
* DESCRIPTION:
* INPUTS:               N/A
* OUTPUTS:
* RETURN VALUE:
***************************************************************************/
void EipInit()
{
```

```
    int (*fun_ptr)(unsigned char*, short,void*);
    fun_ptr = NULL;//MyCallbackFunc;

    /* ---------------- */
    /* open EIP session */
    /* ---------------- */
    open_session_in.cNotifyEvant=1;
    if (EipOpenSession(fun_ptr, &open_session_in, &open_session_out)!=0)
    {
        printf("open session error %d\n",open_session_out.usErrorID);
    }

    /* ------------------------------------------------------------ */
    /* allocate EIP session memory according to configuration file */
    /* ------------------------------------------------------------ */
    strcpy(create_in.cPath,g_cfg_name);
    if (EipCreate(&create_in,&create_out)!=0)
    {
        printf("create session error %d\n",create_out.usErrorID);
    }

    /* -------------------------- */
    /* get device tags references */
    /* -------------------------- */
    strcpy(ref_by_name_in.cName,"ServoCmd");
    EipGetDevTagRefByName(&ref_by_name_in, &ref_by_name_out);
    servo_cmd_ref = ref_by_name_out.usTagRef;

    strcpy(ref_by_name_in.cName,"PosCmd");
    EipGetDevTagRefByName(&ref_by_name_in, &ref_by_name_out);
    pos_cmd_ref = ref_by_name_out.usTagRef;

    /* --------------------------- */
    /* get adapter tags references */
    /* --------------------------- */
    strcpy(ref_by_name_in.cName,"AxisParams");
    EipGetAdpTagRefByName(&ref_by_name_in,&ref_by_name_out);
    axis_data_ref = ref_by_name_out.usTagRef;

    strcpy(ref_by_name_in.cName,"SR_data");
    EipGetAdpTagRefByName(&ref_by_name_in,&ref_by_name_out);
    sr_data_ref = ref_by_name_out.usTagRef;
    printf("sr_data_ref %d\n",sr_data_ref);

    /* Assemblies instance references */
    ref_by_instance_in.iInstance =1;
    EipGetAssemblyRefByInstance (&ref_by_instance_in, &ref_by_instance_out);
    velocity_ref = ref_by_instance_out.usTagRef;

    /*Init position to current actual position */
    MMC_ReadActualPositionCmd(g_conn_hndl,  axis_ref, &read_pos_in,    &read_pos_out);
    position = read_pos_out.dbPosition;
    printf("start position is %f\n",position);

}
```

11.11. DS-401 Drive Communication

This section details the CANbus input and output communication to the G-MAS (DS-401 drive communications). This form of communication uses the CANopen protocol and device profile specification for embedded systems used in automation.

The purpose of Input/Output modules is to connect sensors and actuators to CANopen networks. In operational mode, input data can be transmitted from the inputs via TPDOs. By default, the PDO transmission is triggered by an interrupt (event). Optionally, PDOs may be transmitted synchronously or remotely requested. In addition, it is possible to read input data via SDO communication from another module, or to write data via SDO to the network, if the module provides SDO client functions. Output data can be received via RPDO by those Input/Output modules that have output capabilities. Output data can also be received via SDO communication services. However, the main purpose of SDO communication is to configure an Input/Output module. Via SDO, the module can receive Input/Output configuration data, and parameters for converting data into meaningful measurements and so on. Input/Output modules compliant with this device profile use pre-defined PDOs. The default mapping of application objects into TPDO and respectively RPDO may be changed via SDO, if variable PDO mapping is supported. An Input/Output module may provide optionally Sync producer/consumer, Time-Stamp producer/consumer, and Emergency producer/consumer functions. For new servo driver designs, it is highly recommended to support Heartbeat functions.

11.12. DS-401 Function Blocks

The following DS-401 drive communication function blocks are described:

DS-401 Drive Communication
MMC_CancelGeneralRPDO3
MMC_CancelGeneralRPDO4
MMC_CancelGeneralTPDO3
MMC_CancelGeneralTPDO4
MMC_ConfigGeneralRPDO3
MMC_ConfigGeneralRPDO4
MMC_ConfigGeneralTPDO3
MMC_ConfigGeneralTPDO4
MMC_DisableDIChangedEvent
MMC_EnableDIChangedEvent
MMC_ReadDS401DIGroup
MMC_ReadDS401DInput
MMC_WriteDS401DOGroup
MMC_WriteDS401DOutput

11.12.1. MMC_CancelGeneralRPDO3

Cancels the general configuration of the DS-401 node or G-MAS for RX at PDO3.

```
MMC_LIB_API int MMC_CancelGeneralRPDO3(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CANCELGENERALRPDO3_IN* pInParam,
OUT MMC_CANCELGENERALRPDO3_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CANCELGENERALRPDO3_IN** input data structure using the MMC_CancelGeneralRPDO3 function.

pOutParam

Points to the **MMC_CANCELGENERALRPDO3_OUT** output structure receiving information, as a result of calling the MMC_CancelGeneralRPDO3 function.

Remarks

Cancels communications to read/write PDO's sent from the G-MAS or host.

Scope

All

MMC_CANCELGENERALRPDO3_IN Structure

```
typedef struct{
unsigned char ucDummy;
}MMC_CANCELGENERALRPDO3_IN;
```

Parameters

ucDummy

Dummy values. Any +ve character value.

MMC_CANCELGENERALRPDO3_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CANCELGENERALRPDO3_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 11-87 describes the function block for MMC_CancelGeneralRPDO3

Figure 11-87: MMC_CancelGeneralRPDO3 function block

11.12.1.2. Function Block Code Example

Refer to the example in section 11.12.4.2.

11.12.2. MMC_CancelGeneralRPDO4

Cancels the general configuration of the DS-401 node or G-MAS for RX at PDO4.

```
MMC_LIB_API int MMC_CancelGeneralRPDO4(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CANCELGENERALRPDO4_IN* pInParam,
OUT MMC_CANCELGENERALRPDO4_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CANCELGENERALRPDO4_IN** input data structure using the MMC_CancelGeneralRPDO4 function.

pOutParam

Points to the **MMC_CANCELGENERALRPDO4_OUT** output structure receiving information, as a result of calling the MMC_CancelGeneralRPDO4 function.

Remarks

Cancels communications to read/write PDO's sent from the G-MAS or host.

Scope

All

MMC_CANCELGENERALRPDO4_IN Structure

```
typedef struct{
unsigned char ucDummy;
}MMC_CANCELGENERALRPDO4_IN;
```

Parameters

ucDummy

Dummy values. Any +ve character value.

MMC_CANCELGENERALRPDO4_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CANCELGENERALRPDO4_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 11-88 describes the function block for MMC_CancelGeneralRPDO4

Figure 11-88: MMC_CancelGeneralRPDO4 function block

11.12.2.2. Function Block Code Example

```
MMC_CANCELGENERALRPDO3_IN Cancel3InParam;
MMC_CANCELGENERALRPDO3_OUT Cancel3OutParam;
MMC_CANCELGENERALRPDO4_IN Cancel4InParam;
MMC_CANCELGENERALRPDO4_OUT Cancel4OutParam;

rc = MMC_CancelGeneralRPDO3(ConnHndl,hAxisRef & Cancel3InParam,&Cancel3OutParam);
if(rc != 0)
{
      printf("MMC_CancelGeneralRPDO3 failed, error %d\n",Cancel3OutParam.usErrorID);
}

rc = MMC_CancelGeneralRPDO4(ConnHndl,hAxisRef,&Cancel4InParam,&Cancel4OutParam);
if(rc != 0)
{
      printf("MMC_CancelGeneralRPDO4 failed, error %d\n",Cancel4OutParam.usErrorID);
}
```

11.12.2.3. Implementation Example

11.12.3. MMC_CancelGeneralTPDO3

Cancels the general configuration of the DS-401 node or G-MAS for TX at PDO3.

```
MMC_LIB_API int MMC_CancelGeneralTPDO3(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CANCELGENERALTPDO3_IN* pInParam,
OUT MMC_CANCELGENERALTPDO3_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CANCELGENERALTPDO3_IN** input data structure using the MMC_CancelGeneralTPDO3 function.

pOutParam

Points to the **MMC_CANCELGENERALTPDO3_OUT** output structure receiving information, as a result of calling the MMC_CancelGeneralTPDO3 function.

Remarks

Cancels communications to read/write PDO's sent from the G-MAS or host.

Scope

All

MMC_CANCELGENERALTPDO3_IN Structure

```
typedef struct{
unsigned char ucDummy;
}MMC_CANCELGENERALTPDO3_IN;
```

Parameters

ucDummy

Dummy values. Any +ve character value.

MMC_CANCELGENERALTPDO3_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CANCELGENERALTPDO3_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 11-89 describes the function block for MMC_CancelGeneralTPDO3

Figure 11-89: MMC_CancelGeneralTPDO3 function block

11.12.3.2. Function Block Code Example

Refer to the example in section 11.12.4.2.

11.12.4. MMC_CancelGeneralTPDO4

Cancels the general configuration of the DS-401 node or G-MAS for TX at PDO4.

```
MMC_LIB_API int MMC_CancelGeneralTPDO4(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CANCELGENERALTPDO4_IN* pInParam,
OUT MMC_CANCELGENERALTPDO4_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CANCELGENERALTPDO4_IN** input data structure using the MMC_CancelGeneralTPDO4 function.

pOutParam

Points to the **MMC_CANCELGENERALTPDO4_OUT** output structure receiving information, as a result of calling the MMC_CancelGeneralTPDO4 function.

Remarks

Cancels communications to read/write PDO's sent from the G-MAS or host.

Scope

All

MMC_CANCELGENERALTPDO4_IN Structure

```
typedef struct{
unsigned char ucDummy;
}MMC_CANCELGENERALTPDO4_IN;
```

Parameters

ucDummy

Dummy values. Any +ve character value.

MMC_CANCELGENERALTPDO4_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CANCELGENERALTPDO4_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 11-90 describes the function block for MMC_CancelGeneralTPDO4.

Figure 11-90: MMC_CancelGeneralTPDO4 function block

11.12.4.2. Function Block Code Example

```
MMC_CANCELGENERALTPDO3_IN Cancel3InParam;
MMC_CANCELGENERALTPDO3_OUT Cancel3OutParam;
MMC_CANCELGENERALTPDO4_IN Cancel4InParam;
MMC_CANCELGENERALTPDO4_OUT Cancel4OutParam;

rc = MMC_CancelGeneralTPDO3(ConnHndl,hAxisRef & Cancel3InParam,&Cancel3OutParam);
if(rc != 0)
{
      printf("MMC_CancelGeneralTPDO3 failed, error %d\n",Cancel3OutParam.usErrorID);
}

rc = MMC_CancelGeneralTPDO4(ConnHndl,hAxisRef,&Cancel4InParam,&Cancel4OutParam);
if(rc != 0)
{
      printf("MMC_CancelGeneralTPDO4 failed, error %d\n",Cancel4OutParam.usErrorID);
}
```

11.12.5. MMC_ConfigGeneralRPDO3

Generally configures the DS-401 node or G-MAS for RX at PDO3.

```
MMC_LIB_API int MMC_ConfigGeneralRPDO3(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGGENERALRPDO3_IN* pInParam,
OUT MMC_CONFIGGENERALRPDO3_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGGENERALRPDO3_IN** input data structure using the MMC_ConfigGeneralRPDO3 function.

pOutParam

Points to the **MMC_CONFIGGENERALRPDO3_OUT** output structure receiving information, as a result of calling the MMC_ConfigGeneralRPDO3 function.

Remarks

Opens communications to read/write PDO's sent from the G-MAS or host. Make sure to map the PDOs by himself before using these APIs.

Scope

All

MMC_CONFIGGENERALRPDO3_IN Structure

```
typedef struct{
unsigned char ucEventType;
unsigned char ucPDOCommParam;
}MMC_CONFIGGENERALRPDO3_IN;
```

Parameters

ucEventType

Defines which group of events are to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738** the correct definition to be used. Any +ve character values are acceptable.

ucPDOCommParam

PDO communications parameter. Has the following +ve character values:

PDO_COM_PARAM_SYNC	0x01
PDO_COM_PARAM_ASYNC	0xFF

MMC_CONFIGGENERALRPDO3_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGGENERALRPDO3_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-91 describes the function block for MMC_ConfigGeneralRPDO3

Figure 11-91: MMC_ConfigGeneralRPDO3 function block

11.12.5.2. Function Block Code Example

Refer to the example in section 11.12.6.2.

11.12.6. MMC_ConfigGeneralRPDO4

Generally configures the DS-401 node or G-MAS for RX at PDO4.

```
MMC_LIB_API int MMC_ConfigGeneralRPDO4(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGGENERALRPDO4_IN* pInParam,
OUT MMC_CONFIGGENERALRPDO4_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGGENERALRPDO4_IN** input data structure using the MMC_ConfigGeneralRPDO4 function.

pOutParam

Points to the **MMC_CONFIGGENERALRPDO4_OUT** output structure receiving information, as a result of calling the MMC_ConfigGeneralRPDO4 function.

Remarks

Opens communications to read/write PDO's sent from the G-MAS or host.

Scope

All

MMC_CONFIGGENERALRPDO4_IN Structure

```
typedef struct{
unsigned char ucEventType;
unsigned char ucPDOCommParam;
}MMC_CONFIGGENERALRPDO4_IN;
```

Parameters

ucEventType

Defines which group of events are to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738** the correct definition to be used. Any +ve character values are acceptable.

ucPDOCommParam

PDO communications parameter. Has the following +ve character values:

PDO_COM_PARAM_SYNC	0x01
PDO_COM_PARAM_ASYNC	0xFF
PDO_COM_PARAM_EVENT	0xFE

PDO events are only possible when the input argument *ucPDOCommParam*, is PDO_COM_PARAM_EVENT.

MMC_CONFIGGENERALRPDO4_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGGENERALRPDO4_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-92 describes the function block for MMC_ConfigGeneralRPDO4

Figure 11-92: MMC_ConfigGeneralRPDO4 function block

11.12.6.2. Function Block Code Example

```
      MMC_CONFIGGENERALRPDO3_IN ConfigRPDO3InParam;
      MMC_CONFIGGENERALRPDO3_OUT ConfigRPDO3OutParam;
      MMC_CONFIGGENERALRPDO4_IN ConfigRPDO4InParam;
      MMC_CONFIGGENERALRPDO4_OUT ConfigRPDO4OutParam;

         ConfigRPDO3InParam.ucEventType = 16;
         ConfigRPDO4InParam.ucEventType = 17;
         ConfigRPDO3InParam.ucPDOCommParam = 0x01;
         ConfigRPDO4InParam.ucPDOCommParam = 0x01;
rc = MMC_ConfigGeneralRPDO3(ConnHndl,hAxisRef,& ConfigRPDO3InParam,&ConfigRPDO3OutParam);
      if(rc != 0)
      {
            printf("MMC_ConfigGeneralRPDO3 failed, error %d\n",ConfigRPDOOutParam.usErrorID);
      }

rc = MMC_ConfigGeneralRPDO4(ConnHndl,hAxisRef,&ConfigRPDO4InParam,&ConfigRPDO4OutParam);
      if(rc != 0)
      {
            printf("MMC_ConfigGeneralRPDO4 failed, error %d\n",ConfigRPDO4OutParam.usErrorID);
      }
```

11.12.6.3. Implementation Example

11.12.7. MMC_ConfigGeneralTPDO3

Generally configures the DS-401 node or G-MAS for TX at PDO3.

```
MMC_LIB_API int MMC_ConfigGeneralTPDO3(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGGENERALTPDO3_IN* pInParam,
OUT MMC_CONFIGGENERALTPDO3_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGGENERALTPDO3_IN** input data structure using the MMC_ConfigGeneralTPDO3 function.

pOutParam

Points to the **MMC_CONFIGGENERALTPDO3_OUT** output structure receiving information, as a result of calling the MMC_ConfigGeneralTPDO3 function.

Remarks

Opens communications to read/write PDO's sent from the G-MAS or host.

Scope

All

MMC_CONFIGGENERALTPDO3_IN Structure

```
typedef struct{
unsigned char ucEventType;
}MMC_CONFIGGENERALTPDO3_IN;
```

Parameters

ucEventType

Defines which group of events are to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738** the correct definition to be used. Any +ve character values are acceptable.

MMC_CONFIGGENERALTPDO3_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGGENERALTPDO3_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-93 describes the function block for MMC_ConfigGeneralTPDO3

Figure 11-93: MMC_ConfigGeneralTPDO3 function block

11.12.7.2. Function Block Code Example

Refer to the example in section 11.12.8.2.

11.12.8. MMC_ConfigGeneralTPDO4

Generally configures the DS-401 node or G-MAS for TX at PDO4.

```
MMC_LIB_API int MMC_ConfigGeneralTPDO4(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_CONFIGGENERALTPDO4_IN* pInParam,
OUT MMC_CONFIGGENERALTPDO4_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CONFIGGENERALTPDO4_IN** input data structure using the MMC_ConfigGeneralTPDO4 function.

pOutParam

Points to the **MMC_CONFIGGENERALTPDO4_OUT** output structure receiving information, as a result of calling the MMC_ConfigGeneralTPDO4 function.

Remarks

Opens communications to read/write PDO's sent from the G-MAS or host.

Scope

All

MMC_CONFIGGENERALTPDO4_IN Structure

```
typedef struct{
unsigned char ucEventType;
}MMC_CONFIGGENERALTPDO4_IN;
```

Parameters

ucEventType

Defines which group of events are to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738** the correct definition to be used. Any +ve character values are acceptable.

MMC_CONFIGGENERALTPDO4_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CONFIGGENERALTPDO4_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-94 describes the function block for MMC_ConfigGeneralTPDO4

Figure 11-94: MMC_ConfigGeneralTPDO4 function block

11.12.8.2. Function Block Code Example

```
      MMC_CONFIGGENERALTPDO3_IN ConfigTPDO3InParam;
      MMC_CONFIGGENERALTPDO3_OUT ConfigTPDO3OutParam;
      MMC_CONFIGGENERALTPDO4_IN ConfigTPDO4InParam;
      MMC_CONFIGGENERALTPDO4_OUT ConfigTPDO4OutParam;

     ConfigTPDO3OutParam.ucEventType = 16;
     ConfigTPDO4InParam.ucEventType = 17;
rc = MMC_ConfigGeneralTPDO3(ConnHndl,hAxisRef,& ConfigTPDO3InParam,&ConfigTPDO3OutParam);
      if(rc != 0)
      {
            printf("MMC_ConfigGeneralTPDO3 failed, error %d\n",ConfigTPDO3OutParam.usErrorID);
      }

rc = MMC_ConfigGeneralTPDO4(ConnHndl,hAxisRef,&ConfigTPDO4InParam,&ConfigTPDO4OutParam);
      if(rc != 0)
      {
            printf("MMC_ConfigGeneralTPDO4 failed, error %d\n",ConfigTPDO4OutParam.usErrorID);
      }
```

11.12.9. MMC_DisableDS401DIChangedEvent

Disables a DS401 digital input event change against an I/O module.

```
MMC_LIB_API int MMC_DisableDS401DIChangedEvent(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_DISABLEDICHANGEDEVENT_IN* pInParam,
OUT MMC_DISABLEDICHANGEDEVENT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_DISABLEDICHANGEDEVENT_IN** input data structure using the MMC_DisableDS401DIChangedEvent function.

pOutParam

Points to the **MMC_DISABLEDICHANGEDEVENT_OUT** output structure receiving information, as a result of calling the MMC_DisableDS401DIChangedEvent function.

Remarks

When disabled, prevents any DS401 digital input event change being sent from the I/O module to the G-MAS and then host server (if connected).

Scope

All

MMC_DISABLEDICHANGEDEVENT_IN Structure

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

Parameters

ucDummy

Dummy data input. Any +ve character value.

MMC_DISABLEDICHANGEDEVENT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_DISABLEDICHANGEDEVENT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 11-95 describes the function block for MMC_DisableDS401DIChangedEvent

Figure 11-95: MMC_DisableDS401DIChangedEvent function block

11.12.9.2. Function Block Code Example

```
int rc;
MMC_DISABLEDICHANGEDEVENT_IN        stDisableDIChangeEv_in;
MMC_DISABLEDICHANGEDEVENT_OUT       stDisableDIChangeEv_out;
//
// Inserting the structure parameters:
stDisableDIChangeEv_in.ucDummy  = 1;     //Dummy data input
//
rc = MMC_DisableDS401DIChangedEvent (hConn, iAxisRef, &stDisableDIChangeEv_in,
&stDisableDIChangeEv_out);
if (rc != 0)
{
    HandleError();
}
```

11.12.10. MMC_EnableDS401DIChangedEvent

Enables an DS401 digital input event change.

```
MMC_LIB_API int MMC_EnableDS401DIChangedEvent (
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_ENABLEDICHANGEDEVENT_IN* pInParam,
OUT MMC_ENABLEDICHANGEDEVENT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_ENABLEDICHANGEDEVENT_IN** input data structure using the MMC_EnableDS401DIChangedEvent function.

pOutParam

Points to the **MMC_ENABLEDICHANGEDEVENT_OUT** output structure receiving information, as a result of calling the MMC_EnableDS401DIChangedEvent function.

Remarks

When enabled, any DS401 digital input event change is sent from the I/O module to the G-MAS and then host server (if connected).

Scope

All

MMC_ENABLEDICHANGEDEVENT_IN Structure

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

Parameters

ucDummy

Dummy data input. Any +ve character value.

MMC_ENABLEDICHANGEDEVENT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_ENABLEDICHANGEDEVENT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 11-96 describes the function block for MMC_EnableDS401DIChangedEvent

Figure 11-96: MMC_EnableDS401DIChangedEvent function block

11.12.10.2. Function Block Code Example

```
int rc;
MMC_ENABLEDICHANGEDEVENT_IN         stEnableDIChangeEv_in;
MMC_ENABLEDICHANGEDEVENT_OUT        stEnableDIChangeEv_out;
//
// Inserting the structure parameters:
stEnableDIChangeEv_in.ucDummy   = 1;     //Dummy data input
//
rc = MMC_EnableDS401DIChangedEvent (hConn, iAxisRef, &stEnableDIChangeEv_in,
&stEnableDIChangeEv_out);
if (rc != 0)
{
    HandleError();
}
```

11.12.10.3. Implementation Example

11.12.11. MMC_ReadDS401DIGroup

Reads the DS-401 digital inputs of a group of 8 digital I/Os.

```
MMC_LIB_API int MMC_ReadDS401DIGroup(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_READDIGROUP_IN* pInParam,
OUT MMC_READDIGROUP_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READDIGROUP_IN** input data structure using the MMC_ReadDIGroup function.

pOutParam

Points to the **MMC_READDIGROUP_OUT** output structure receiving information, as a result of calling the MMC_ReadDIGroup function.

Remarks

A group consists of 8 I/O connections with a possible maximum of 8 groups and therefore 64 I/O connections.

Scope

All

MMC_READDIGROUP_IN Structure

```
typedef struct{
unsigned char ucGroupIndex;
}MMC_READDIGROUP_IN;
```

Parameters

ucGroupIndex

Group index of 8 I/O's up to a max of 64 I/O's. +ve Integer (character) values of [9 onwards]

Note that It is possible to write to the lower 8 groups (64 bits) using the function MMC_WriteDS401DOutput that writes to the first 64 bits (long long variable) if they are valid. MMC_WriteDS401DOGroup only writes to the upper bits.

MMC_READDIGROUP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_READDIGROUP_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-97 describes the function block for MMC_Read DS401DIGroup

Figure 11-97: MMC_ReadDS401DIGroup function block

11.12.11.2. Function Block Code Example

```
int rc;
MMC_READDIGROUP_IN          stReadDIGroup_in;
MMC_READDIGROUP_OUT         stReadDIGroup_out;
//
// Inserting the structure parameters:
stReadDIGroup_in.ucGroupIndex   = 21;         //Group index
//
rc = MMC_ReadDS401DIGroup (hConn, iAxisRef, &stReadDIGroup_in, &stReadDIGroup_out);
printf("ErrId[%d]\n", (short)stReadDIGroup_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.12.12. MMC_ReadDS401DInput

Reads the DS-401 digital input of all 64 bit I/O's in one action, increasing the communication speed proportionately versus reading 8 x groups of 8 I/O's.

```
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
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_READDI_IN** input data structure using the MMC_ReadDS401DInput function.

pOutParam

Points to the **MMC_READDI_OUT** output structure receiving information, as a result of calling the MMC_ReadDS401DInput function.

Remarks

None

Scope

All

MMC_READDI_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_READDI_IN;
```

Parameters

dummy

Dummy data input. Any +ve character value.

MMC_READDI_OUT Structure

```
typedef struct{
#ifdef WIN32
unsigned __int64 ulliDI;
#else
unsigned long long int ulliDI;
#endif
unsigned short usStatus;
short usErrorID;
}MMC_READDI_OUT;
```

Parameters

__int64 ulliDI or ulliDI

If function is defined for WIN32 then use *__int64 ulliDI*, else use *ulliDI*. Any +ve, –ve (Win32) or +ve 64bit (8 bytes) character and/or integer.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-98 describes the function block for MMC_ReadDS401DInput

Figure 11-98: MMC_ReadDS401DInput function block

11.12.12.2. Function Block Code Example

```
int rc;
MMC_READDI_IN      stReadDI_in;
MMC_READDI_OUT     stReadDI_out;
//
// Inserting the structure parameters:
stReadDI_in.dummy      = 1;      //dummy input
//
rc = MMC_ReadDS401DInput (hConn, iAxisRef, &stReadDI_in, &stReadDI_out);
printf("DS-401 Input Status[%ld] ErrId[%d]\n", (long int)stReadDI_out.ulliDI,
(short)stReadDI_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

11.12.13. MMC_WriteDS401DOGroup

Writes the DS-401 digital outputs of a group of 8 I/O's to the G-MAS.

```
MMC_LIB_API int MMC_WriteDS401DOGroup(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_WRITEDOGROUP_IN* pInParam,
OUT MMC_WRITEDOGROUP_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEDOGROUP_IN** input data structure using the MMC_WriteDS401DOGroup function.

pOutParam

Points to the **MMC_WRITEDOGROUP_OUT** output structure receiving information, as a result of calling the MMC_WriteDS401DOGroup function.

Remarks

A group consists of 8 I/O connections with a possible maximum of 8 groups and therefore 64 I/O connections.

Scope

All

MMC_WRITEDOGROUP_IN Structure

```
typedef struct{
unsigned char ucGroupIndex;
unsigned char ucVal;
}MMC_WRITEDOGROUP_IN;
```

Parameters

ucGroupIndex

Group index of 8 I/O's up to a max of 64 I/O's. +ve Integer (character) values of [9 onwards]

Note that It is possible to write to the lower 8 groups (64 bits) using the function MMC_WriteDS401DOutput that writes to the first 64 bits (long long variable) if they are valid. MMC_WriteDS401DOGroup only writes to the upper bits.

ucVal

Digital output value of the 0 – 8 bit data in a group, ranging from 0 – 255. Any +ve character value.

MMC_WRITEDOGROUP_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEDOGROUP_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-99 describes the function block for MMC_WriteDS401DOGroup

Figure 11-99: MMC_WriteDS401DOGroup function block

11.12.13.2. Function Block Code Example

```
int rc;
MMC_WRITEDOGROUP_IN        stWriteDOGroup_in;
MMC_WRITEDOGROUP_OUT       stWriteDOGroup_out;
//
// Inserting the structure parameters:
stWriteDOGroup_in.ucGroupIndex  = 10;        //Index of the group axes
stWriteDOGroup_in.ucVal         = 9;         //Digital output value
//
rc = MMC_WriteDS401DOGroup (hConn, iAxisRef, &stWriteDOGroup_in, &stWriteDOGroup_out);
if (rc != 0)
{
    HandleError();
}
```

11.12.13.3. Implementation Example

11.12.14. MMC_WriteDS401DOutput

Writes to all of the DS-401 digital outputs assigned to TPDO1 at once, up to 64 bit I/O's in one action, increasing the communication speed proportionately versus writing to 8 x groups of 8 I/O's.

```
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
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_DS401_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_WRITEDO_IN** input data structure using the MMC_WriteDS401DOutput function.

pOutParam

Points to the **MMC_WRITEDO_OUT** output structure receiving information, as a result of calling the MMC_WriteDS401DOutput function.

Remarks

None

Scope

All

MMC_WRITEDO_IN Structure

```
typedef struct{
#ifdef WIN32
unsigned __int64 ulliDO;
#else
unsigned long long int ulliDO;
#endif
}MMC_WRITEDO_IN;
```

Parameters

__int64 ulliDO or ulliDO

If function is defined for WIN32 then use *__int64 ulliDO*, else use *ulliDO*. Any +ve, –ve (Win32) or +ve 64bit (8 bytes) character and/or integer.

MMC_WRITEDO_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_WRITEDO_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 11-100 describes the function block for MMC_WriteDS401DOutput

Figure 11-100: MMC_WriteDS401DOutput function block

11.12.14.2. Function Block Code Example

```
int rc;
MMC_WRITEDO_IN          stWriteDO_in;
MMC_WRITEDO_OUT         stWriteDO_out;
//
// Inserting the structure parameters:
stWriteDO_in.ulliDO    = 1;     //Value to write to digital outputs
//
rc = MMC_WriteDS401DOutput (hConn, iAxisRef, &stWriteDO_in, &stWriteDO_out);
if (rc != 0)
{
    HandleError();
}
```

Chapter 12: API Events

Event handling in the G-MAS is the ability to capture specific events occurring within the G-MAS, and send Asynchronous Events Callback messages to a host, thus mirroring the occurrences of the event. The mechanism to handle events in the G-MAS involves the following:

- Communication, async replies from the drive, e.g. the function block MMC_SendSDO
- Process progress, notifies the host regarding the progress of a long ongoing process, such as Download Firmware
- Errors in the drive notifications, per node
- PDO3 and PDO4 receive, per node
- System Errors – General system failures (Not yet implemented)
- On Motion End, per node
- On Heartbeat Error, per node
- Emergencies, per node
- Modbus writes from hosts
- Touch Probe event received
- Node Errors
- Axis stopped due to limit
- PVT Underflow data warning
- CAN Node returning to network
- CAN ASYNC reply from drive is ready to be read

This chapter describes the situations when such an event is triggered, the data structure and format of each event.

The treatment of an event is per connection, in the open UDP port. The UDP port is automatically opened by the G-MAS function when the MMC_InitConnection function is called and MMC_OpenUdpChannel is invoked. If the MMC_InitConnection and MMC_OpenUdpChannel functions were called with a callback function that is a valid pointer to a callback function, a UDP listening port is automatically created. This UDP port listens for incoming messages, actually on a thread. Once a message is received, the registered callback function is called.

After the MMC_InitConnection function is called, the GMAS_OpenUDPResponceChannel function is called. This function sends the previously opened appropriate port, the command MMC_InitConnection, to the G-MAS. In addition to the port, a 32-bit variable is sent, stating the event types that are to be registered in the G-MAS.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
														TABLE_UNDERFLOW_EVT	STOP_ON_LIMIT_EVT	NODE_ERROR_EVT	TOUCH_PROBE_ENDED_EVT	MODBUS_WRITE_EVT	SYSTEMERROR_EVT	HOME_ENDED_EVT	EMIT_EVT	DRVERROR_EVT	PDORCV_EVT	HBEAT_EVT	MOTIONENDED_EVT	EMCY_EVT	DOWNLOAD_FW_EVT				ASYNC_REPLY_EVT

Figure 12-1: 32-bit variable

The user sends the appropriate events he wishes to receive as a callback event. There are situations when stating the event is insufficient, and additional function calls must be made. This will be discussed per event type.

12.1. Communication Byte Order

The user must recognize the value of the endianness (little/big endian) of the system on which his program is executed. The message structure of asynchronous call-back buffer, as described hereby refers to data location (offset) of different types. The user may have to convert the endianness, if his program runs on a Windows system.

12.2. Communication ASYNC Replies (Events) From Drives

This event is received if an error (or timeout) occurred when calling an SDO download or Drive Command via the binary interpreter mechanism. Please note that in this situation, unlike other events, the Axis reference, is taken from offset 14 (instead of 12).

The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
0	ASYNC_REPLY_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	14	2	
		Status	Unsigned short	8	2	
		Error ID	Unsigned short	10	2	
		COB ID	Unsigned short	12	2	
		Length	Unsigned char	16	1	Length in bytes
		Data	Unsigned char	17	According to Length	
		Async Event Type	Unsigned char	25	1	

When an asynchronous operation is successful, and this is a response for an inquiry operation (similar to the binary interpreter command "get" which sends a raw data operation or SDO upload request) the status and error ID fields will hold the value of zero, and also data is returned in the data field.

When the operation fails, the status field holds value other than zero and the error code field states a specific error code which defines the occurrence during the operation. If the error occurred during an SDO upload or dDownload, the data field will hold the full CAN message returned from the node.

The Async Event Type field holds values from the Async events numerators group found in the MMC_events_API.h file. The result of these asynchronous operational events are used to indicate the following:

- SendSDO operations (upload \ download)
- Motion mode change
- Binary interpreter operations
- Read DI\write DO
- PDO3\4 operations (mapping, canceling or choosing communication parameter)
- Reset node operation
- Virtual encoder configuration
- Bulk upload

Some of these event results are indicated only when an error occurs (no ACK response). For a binary interpreter operation like "get" no response is sent (timeout is simulated).

12.3. Download Firmware Notifications

This event is received to update the progress of an ongoing download firmware procedure. Please contact to Elmo's representative for further support.

12.4. Emergency Event

Emergency event is triggered by the occurrence of a CANopen device internal error situation.

The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
5	EMCY_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Emergency Error Code	Unsigned short	14	2	As per CiA DS301 (chapter 9.2.5.3)
		MSEF	Unsigned char	16	5	Manufacturer-specific error code
		ER	Unsigned char	15	1	Error register

12.5. Motion Ended Event

Motion ended event is triggered by GMAS when an axis (single as well as group axis) motion is ended. The user must register for this event in order to receive event notification and in addition he must call "MMC_EnableMotionEndedEventCmd" API and specify the particular axis from which he wishes to receive Motion Ended Events. The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
6	MOTIONENDED_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Error ID	short	10	2	

12.6. Heart Beat Event

Heart Beat error event is triggered by GMAS when no heartbeat event was received by GMAS from one of the devices.

The UDP Data received is as follows:

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

12.7. PDO Receive Event

This event is triggered by the GMAS when a PDO (3 or 4) for a specific axis is received, and was configured to be sent as notification to the user. Note that the term of the variable located at offset 14 (see below) refers to the Event Group /PDO Number when handling DS 402/401 respectively. The definition of User Data 1 and User Data 2 is Data which is a 32bit float for event groups 15,16.

The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
8	PDORCV_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Event Group (PDO No. if referring to DS-401) with possible values of 1-17 (Refer to next subsections)	Unsigned char	14	2	Event Group or PDO No. is assigned by the values 1 – 17 and the remaining UDP data is arranged according to the value

12.7.1. Event Group equals to 5 or 6

The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
8	PDORCV_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Event Group	Unsigned char	14	2	= 5 or 6
		User Data 1	short	15	2	Result must be multiplied by rated current.
		User Data 2	long	17	4	

12.7.2. Event Group equals to 11

The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
8	PDORCV_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Event Group	Unsigned char	14	2	= 11
		User Data	Long	15	4	

12.7.3. Event Group Equals to 16 or 17

The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
8	PDORCV_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Event Group	Unsigned char	14	2	= 16 or 17
		User Data	Long long	15	8	

12.7.4. Event Group equals to 1 – 15 besides 5, 6, 11, 16 and 17

The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
8	PDORCV_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Event Group	Unsigned char	14	2	= 1-15 excluding 5,6,11
		User Data 1	Long	15	4	Data is 32bit float for group 15,16.
		User Data 2	long	19	4	

12.8. Home Ended Event

This event is triggered by the GMAS when a Single axis finished the homing procedure. The UDP Data received is as follows:

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

12.9. Modbus Write Event

Modbus write event is triggered by GMAS when user writes to Modbus Holding registers. The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
13	MODBUS_WRITE_EVT	Event No.	Unsigned short	0	2	

12.10. Touch Probe Ended Event

Touch Probe event is triggered by GMAS when a touch probe position is received from one of the drives on the network. This is relevant to Ethercat drives only, and drives must be configured appropriately. The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
14	TOUCH_PROBE_ENDED_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Touch probe Position	long	14	2	

12.11. Node Connected Event

Node connected event is triggered by GMAS when an axis is re-connected to the network. This event will occur in two possible situations:

- 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 UDP 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	

12.12. Node Error Event

Node error event is triggered when error occurs on one of axes (Single / Group). The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
15	NODE_ERROR_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Error ID	Unsigned short	10	2	
		Emergency Code	Unsigned short	14	2	

12.13. Stop ON Limit Event

This event is received when axes (single or group) stopped on a software /hardware limit. The UDP Data received is as follows:

Event No.	Event Constant Definition	Data				Comment
		Name	Type	Offset	Length (bytes)	
16	STOP_ON_LIMIT_EVT	Event No.	Unsigned short	0	2	
		Axis Ref	Unsigned short	12	2	
		Error ID	Short	10	2	
		Status	Unsigned integer	14	4	
		MSC Limit	Unsigned integer	18	4	

12.14. Table Underflow Event

This event is received when the number of remaining PVT points in the table to be executed by the profiler has reached the minimal limit that was set by the user with MMC_InitTableCmd.

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

12.15. Global Async Reply Event

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

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 in the MMC_events_API.h header file.

12.16. Ethernet-IP Event

This event is received when an EthernetIP communication ends. The event indicates the result of this operation. When in error, the GMAS error code is returned and the status field is different than zero. When succesful, the status and error fields are zero.

Event No.	Event Constant Definition	Data				Comment
		Name	**Type**	**Offset**	**Length (bytes)**	
20	EIP_EVENT	Event No.	Unsigned short	0	2	
		Event ID	Unsigned short	2	2	Events from Ethernet-IP library

12.17. Communication Event Mechanism

Figure 12-2: Communication events mechanism

Figure 12-2 describes the communication event mechanism in the G-MAS. The HPT thread in the kernel is scheduled every basic cycle time and is responsible for populating the Events FIFO in the kernel, when a specific event condition is fulfilled. In the user space (other side), there is a Cycle Time Event thread which awakens every time a new event is entered to the FIFO. This thread processes the new event and sends a UDP message to the registered user on the event. The Cycle Time Event thread is only involved with cycle time based events, i.e. events that are generated in the HPT thread scheduling period.

Another thread in the user space, involving event processing, is the Immediate Event thread. This is responsible for processing events, which require immediate delivery to the registered user. When a condition for an event is fulfilled, at the CAN driver level, the event's data is sent to a special FIFO in the CAN driver. The Immediate Event thread is awakened, when a new event is added to the FIFO. The event is processed and sent via UDP message to the registered user.

It should be noted that the default event mode is 0, No notification to user (default). Therefore, if the event mode is not properly set to cyclic or immediate notification, no event notification will be received.

There are two advantages of the Immediate Event mechanism relative to the Cycle Time Event mechanism:

- Immediate and fast event notification to the user. No need to wait until the next cycle time in order to deliver the event to the user.
- Events can be missed in the Cycle Time Event mechanism, e.g. If two input changes occurred between one cycle and the next, only the last input change event will be delivered to the user, the first input change event is lost. In the Immediate Event mechanism, the above phenomenon does not occur.

12.18. Events Mask and Enumeration

The following lists the Events enumerator and their specific ID, relevant to the parameter *iEventsMask* in C, and *iEventMask* in C++. These events are therefore additionally applicable to C++ events and error states.

Events Mask	Value	Events Enumeration	ID
eEVENT_TYPE_COMM_ASYNC_REPLY	(1 << 0)	ASYNC_REPLY_EVT	0
eEVENT_TYPE_DOWNLOAD_FW	(1 << 4)	DOWNLOAD_FW_EVT	4
eEVENT_TYPE_COMM_EMGCY	(1 << 5)	EMCY_EVT	5
eEVENT_TYPE_COMM_MOTION_ENDED	(1 << 6)	MOTIONENDED_EVT	6
eEVENT_TYPE_COMM_HEARTBEAT_ERROR	(1 << 7)	HBEAT_EVT	7
eEVENT_TYPE_COMM_PDO_RECEIVED	(1 << 8)	PDORCV_EVT	8
eEVENT_TYPE_DRIVE_ERROR	(1 << 9)	DRVERROR_EVT	9
eEVENT_TYPE_EMIT	(1 << 10)	EMIT_EVT	10
eEVENT_TYPE_HOME_ENDED	(1 << 11)	HOME_ENDED_EVT	11
eEVENT_TYPE_SYSTEM_ERROR	(1 << 12)	SYSTEMERROR_EVT	12
eEVENT_TYPE_MODBUS	(1 << 13)	MODBUS_WRITE_EVT	13
eEVENT_TYPE_TOUCH_PROBE_ENDED	(1 << 14)	TOUCH_PROBE_ENDED_EVT	14
eEVENT_TYPE_NODE_ERROR	(1 << 15)	NODE_ERROR_EVT	15
eEVENT_TYPE_STOP_ON_LIMIT	(1 << 16)	STOP_ON_LIMIT_EVT	16
eEVENT_TYPE_TABLE_UNDERFLOW	(1 << 17)	TABLE_UNDERFLOW_EVT	17
eEVENT_TYPE_SEND_ASYNC_EVENT	(1 << 18)		
eEVENT_TYPE_NODE_CONNECTED	(1 << 19)	NODE_CONNECTED_EVT	18
eEVENT_TYPE_GLOBAL_ASYNC_REPLY	(1 << 20)	GLOBAL_ASYNC_REPLY_EVT	19

12.19. Asynchronous Events Callback

This section describes the buffer data structure (pBuff character), which is provided by the G-MAS together with the Event enumerator described in section 12.18 above, as the first input parameter to the callback function. To use asynchronous events callback, it is necessary to implement the mechanism for Callback registration (via MMC_IPCInitConnection, or MMC_RPCInitConnection) and PDO configuration. The user must pre-configure the system to receive Callbacks, e.g. to perform Homing or motion functions, configure PDO's etc.

In addition to an async reply event being received on an error regardless of the event mask, the SEND_ASYNC_EVENT event mask is set. This event triggers the callback function even when the async function has completed successfully (error = 0).

12.19.1. Callback Prototype

The Type definition (MMC_CB_FUNC) is part of the MMC_Definitions.h header file, which also contains the integer CallbackFunc(unsigned char* pBuff, short sSize, void* pIPSock) function.

pBuff

Buffer consisting of all data related to a specific event

sSize

Buffer size (in bytes). +ve numeric format.

pIPSock

The IP socket used. This should not be changed or edited.

12.19.2. Data Structure

Each event sends a buffer with a different data structure. To extract the relevant data from the buffer according to the Event Type, use the following definitions. These definitions are only present in the CPP header file MMCConnection.h.

12.19.3. Event Extraction Example

This user implementation is only for Illustration purposes, but describes specifically how the *pBuff* data in the *xCallbackFunc* function is extracted according to the Event Type, which describes where the data is situated and exactly what the data consists of. For the various local Windows, or .NET select the NetToLocal and endian_swaps below.

The following offset alias table will help in the understanding of the code described below.

Offset Alias	Index
EVENT_ID_INDX	0
AXIS_REF_INDX	12

Offset Alias	Index
ASYNC_EVENT_LEN_INDX	14
PDO_GROUP_INDX	14
PDO_DATA_INDX	15
MSG_DATA_INDX	8
EMGCY_LEN_INDX	4
EMGCY_DATA_INDX	14
TOUCHP_POS_INDX	14

User implementation function Events ID and other type definitions will compile correctly since they emply the same user MMC_DEFINITIONS.H header file.

```
int UserImplementation(unsigned short usAxisRef,...) {/*do yours*/}
int xCallbackFunc(unsigned char* pBuff, short sSize, void* pIPSock) {
        unsigned short usAxisRef;
        unsigned short usStatus;
        unsigned short usErrorId;
        unsigned short usEventID;
        unsigned short usCobID;
        unsigned short usDataLen;
        unsigned short usEventGrp = 0;
        unsigned short usEmergencyCode;
        unsigned long ulData1;
        unsigned long ulData2;
        unsigned char ucEventType;
        MMC_CAN_REPLY_DATA_OUT* pCanReply;
        NetToLocal((void *)&pBuff[EVENT_ID_INDX], &usEventID);
        usAxisRef = (unsigned short)(*(unsigned short*)(&pBuff[AXIS_REF_INDX]));
#ifdef WIN32
endian_swap16(&(usAxisRef));
#endif
switch(usEventID)
{
case ASYNC_REPLY_EVT:
            pCanReply = (MMC_CAN_REPLY_DATA_OUT*)pBuff;
            usStatus = pCanReply->usStatus;
            usErrorId = pCanReply->usErrorid;
```

```
                usCobID = pCanReply->usCOB_ID;
                usAxisRef = pCanReply->usAxisRef;
#ifdef WIN32
            endian_swap16(&(usStatus));
            endian_swap16(&(usErrorId));
            endian_swap16(&(usCobID));
#endif
            usDataLen = pBuff[ASYNC_EVENT_LEN_INDX];
            UserImplementation(usAxisRef, usStatus, usErrorId, usCobID, usDataLen, pBuff);
break ;
case EMCY_EVT:
                UserImplementation(
usAxisRef, (*(unsigned short *)&pBuff[EMGCY_DATA_INDX])); //send axis ref end
    emergency code
break ;
case MOTIONENDED_EVT:
                NetToLocal((void *)&pBuff[MSG_DATA_INDX+2], &usErrorId);
                UserImplementation(usAxisRef, usErrorId == 0); //send axis ref end OK or not
break;
case HBEAT_EVT:
                UserImplementation(usAxisRef); //send axis ref
break ;
case PDORCV_EVT:
                usEventGrp = pBuff[PDO_GROUP_INDX];
    switch (usEventGrp)
{
            case 1:
            case 2:
            case 3:
            case 4:
            case 7:
            case 8:
            case 9:
            case 10:
            case 12:
            case 13:
            case 14:
            case 15:
    NetToLocal((void *)&pBuff[PDO_DATA_INDX], (void *)&ulData1); //type casting as needed
```

```
        NetToLocal((void *)&pBuff[PDO_DATA_INDX+4], (void *)&ulData2); //type casting as needed
        break;
            case 5:
            case 6:
            NetToLocal((void *)&pBuff[PDO_DATA_INDX], &ulData1); //type casting as needed
            NetToLocal((void *)&pBuff[PDO_DATA_INDX+2], (void *)&ulData2); //type casting as needed
        break;
            case 11:
                NetToLocal((void *)&pBuff[PDO_DATA_INDX], (void *)&ulData1);
            ulData2 = 0; //irrelevance
        break;
            default:
        break;
}
        UserImplementation(usAxisRef, usEventGrp, ulData1, ulData2);
        break ;
            case HOME_ENDED_EVT:
                NetToLocal((void *)&pBuff[MSG_DATA_INDX+2], &usErrorId);
                UserImplementation(usAxisRef, usErrorId); //send axis ref and error
        break ;
            case MODBUS_WRITE_EVT:
            //<UserImplementation();>
        break ;
            case TOUCH_PROBE_ENDED_EVT:
                UserImplementation( usAxisRef, *((long*)(&pBuff[TOUCHP_POS_INDX])));
        break ;
case NODE_ERROR_EVT:
                NetToLocal((void *)&pBuff[MSG_DATA_INDX+2], &usErrorId);
                NetToLocal((void *)&pBuff[MSG_DATA_INDX+6], &usEmergencyCode);
                UserImplementation(usAxisRef,usErrorId,usEmergencyCode); //send axis ref end OK or
            not
        break ;
default:
        break ;
            case STOP_ON_LIMIT_EVT:
            TBD
        break ;
case TABLE_UNDERFLOW_EVT:
TBD
```

```
        break ;
case NODE_CONNECTED_EVT:
TBD
        break ;
            case GLOBAL_ASYNC_REPLY_EVT:
            unsigned char ucFuncID = *((unsigned char*)&buffer[12]);
            UserImplementation(ucFuncID,usErrorId,usStatus);
        break ;
            int fnCallback(unsigned char* ucBuffer, short sReqID, void* pSock)
{
        unsigned char ucEventID = ucBuffer[2];
        if (ucBuffer[0] == 20) //Event No. is EIP_EVENT
        switch (ucEventID)
        {
case NM_REQUEST_RESPONSE_RECEIVED:
//      printf("NM_REQUEST_RESPONSE_RECEIVED: sReqID = %d\n", sReqID);
                break;
            case NM_ASSEMBLY_NEW_INSTANCE_DATA:
//      printf("NM_ASSEMBLY_NEW_INSTANCE_DATA: assembly instance = %d\n", sReqID);
                break;
            case NM_ASSEMBLY_NEW_MEMBER_DATA:
                //New data received for the specified assembly member. sReqID contains assembly instance.
                printf("NM_ASSEMBLY_NEW_MEMBER_DATA: assembly instance = %dn", sReqID;
                break;
            case NM_REQUEST_FAILED_INVALID_NETWORK_PATH:
                break;
            case NM_REQUEST_TIMED_OUT:
        printf("NM_REQUEST_TIMED_OUT: sReqID = %d\n", sReqID);
                break;
            case NM_CONNECTION_ESTABLISHED:
//              printf("New connection opened with instance %d\n", sReqID);
                break;
            case NM_CONNECTION_VERIFICATION:
                printf("NM_CONNECTION_VERIFICATION\n");
                break;
            case NM_CONNECTION_RECONFIGURED:
        printf("NM_CONNECTION_RECONFIGURED\n");
        break;
```

```
            case NM_CONNECTION_TIMED_OUT:
//                  printf("Connection with instance %d timed out\n", sReqID);
                    break;
            case NM_CONNECTION_CLOSED:
            //printf("Connection with instance %d closed\n", sReqID);
                    break;
            case NM_CLIENT_OBJECT_REQUEST_RECEIVED:
                    printf("NM_CLIENT_OBJECT_REQUEST_RECEIVED\n");
                    break;
            case NM_PENDING_REQUESTS_LIMIT_REACHED:
                    printf("NM_PENDING_REQUESTS_LIMIT_REACHED\n");
                    break;
            default:
                    printf("%s Unhandled(unknown) response event. %d\n", __func__, ucEventID);
                    break;
        }
        return 0;
}
```

12.19.4. Net To local Conversion

The following code was extricated from the C++ library, and is displayed here to describe the principle. It is however, necessary to create a customized implementation for these functions/macros.

```
inline void NetToLocal(void* NetBuff, unsigned short *usVal)
{
            memcpy((unsigned char*)usVal,(unsigned char*)NetBuff, 2);
            #ifdef WIN32
                    endian_swap16((unsigned short *)usVal);
            #endif
}
inline void NetToLocal(void* NetBuff, void *iVal)
{
            memcpy((unsigned char*)iVal,(unsigned char*)NetBuff,4);
            #ifdef WIN32
                    endian_swap32((unsigned int *)iVal);
            #endif
}
inline void endian_swap16(unsigned short* x)
{*x = (*x>>8) | (*x<<8);}
inline void endian_swap32(unsigned int* x)
```

```
{*x = (*x>>24) | ((*x<<8) & 0x00FF0000) |((*x>>8) & 0x0000FF00) | (*x<<24);
}
```

12.20. Events Function Blocks

The following events function blocks are described:

Events
MMC_ClearEventsMask
MMC_DisableMotionEndedEvent
MMC_EnableMotionEndedEvent
MMC_GetEventsMask
MMC_SetEventsMask

12.20.1. MMC_ClearEventsMask

Clears the events mask for a specific connection depending to the input mask.

```
MMC_LIB_API int MMC_ClearEventsMaskCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_CLEAREVENTSMASK_IN* pInParam,
OUT MMC_CLEAREVENTSMASK_OUT* pOutParam);
);
```

Motion Mode	NC – Supported	Distributed - Supported
Source	GMAS\includes\MMC_events_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_CLEAREVENTSMASK_IN** input data structure using the MMC_ClearEventsMask function.

pOutParam

Points to the **MMC_CLEAREVENTSMASK_OUT** output structure receiving information as a result of calling the MMC_ClearEventsMask function.

Remarks

This involves zeroing the final event data integer of a 32-bit event data with the result of removing the event from the G-MAS. The event to be cleared will depend on the ID event input of the parameter *iEventsMask*.

Scope

All

MMC_CLEAREVENTSMASK_IN Structure

```
typedef struct{
int iEventsMask;
}MMC_CLEAREVENTSMASK_IN;
```

Parameters

iEventsMask

Defined according to the event IDs described in the section **12.18 Events Enumeration on page 12-1010**. Bitwise +ve integer ID.

MMC_CLEAREVENTSMASK_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CLEAREVENTSMASK_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 12-3 describes the function block for MMC_ClearEventsMask

Figure 12-3: MMC_ClearEventsMask function block

12.20.1.2. Function Block Code Example

```
int rc;
MMC_CLEAREVENTSMASK_IN     stClearEventsMask_in;
MMC_CLEAREVENTSMASK_OUT    stClearEventsMask_out;
//
// Inserting the structure parameters:
stClearEventsMask_in.iEventsMask        = 64; //Events mask ID 7 bit 7 is on 1000000 = 64(Dec)
//
rc = MMC_ClearEventsMaskCmd (hConn, &stClearEventsMask_in, &stClearEventsMask_out);
if (rc != 0)
{
    HandleError();
}
```

12.20.2. MMC_DisableMotionEndedEvent

Disables the motion ended event mechanism for a specific node, and no feedback is sent from the G-MAS regarding the progress of the motion.

```
MMC_LIB_API int MMC_DisableMotionEndedEventCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_DISABLEMOTIONENDEDEVENT_IN* pInParam,
OUT MMC_DISABLEMOTIONENDEDEVENT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_events_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_DISABLEMOTIONENDEDEVENT_IN** input data structure using the MMC_DisableMotionEndedEvent function.

pOutParam

Points to the **MMC_DISABLEMOTIONENDEDEVENT_OUT** output structure receiving information, as a result of calling the MMC_DisableMotionEndedEvent function.

Remarks

None

Scope

All

MMC_DISABLEMOTIONENDEDEVENT_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_DISABLEMOTIONENDEDEVENT_IN;
```

Parameters

dummy

Dummy input. Any +ve character value.

MMC_DISABLEMOTIONENDEDEVENT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_DISABLEMOTIONENDEDEVENT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 12-4 describes the function block for MMC_DisableMotionEndedEvent

Figure 12-4: MMC_DisableMotionEndedEvent function block

12.20.2.2. Function Block Code Example

```
int rc;
MMC_DISABLEMOTIONENDEDEVENT_IN      stDisableMotionEndEvent_in;
MMC_DISABLEMOTIONENDEDEVENT_OUT     stDisableMotionEndEvent_out;
//
// Inserting the structure parameters:
stDisableMotionEndEvent_in.dummy        = 1;     //Dummy input
//
rc = MMC_DisableMotionEndedEventCmd (hConn, iAxisRef, &stDisableMotionEndEvent_in,
&stDisableMotionEndEvent_out);
if (rc != 0)
{
    HandleError();
}
```

12.20.3. MMC_EnableMotionEndedEvent

Enables the motion ended event mechanism for a specific node.

```
MMC_LIB_API int MMC_EnableMotionEndedEventCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_ENABLEMOTIONENDEDEVENT_IN* pInParam,
OUT MMC_ENABLEMOTIONENDEDEVENT_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_events_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_ENABLEMOTIONENDEDEVENT_IN** input data structure using the MMC_EnableMotionEndedEvent function.

pOutParam

Points to the **MMC_ENABLEMOTIONENDEDEVENT_OUT** output structure receiving information, as a result of calling the MMC_EnableMotionEndedEvent function.

Remarks

Events are sent regularly from the G-MAS regarding the status of the motion.

Scope

All

MMC_ENABLEMOTIONENDEDEVENT_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_ENABLEMOTIONENDEDEVENT_IN;
```

Parameters

dummy

Dummy input. Any +ve character value.

MMC_ENABLEMOTIONENDEDEVENT_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_ENABLEMOTIONENDEDEVENT_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 12-5 describes the function block for MMC_EnableMotionEndedEvent

Figure 12-5: MMC_EnableMotionEndedEvent function block

12.20.3.2. Function Block Code Example

```
int rc;
MMC_ENABLEMOTIONENDEDEVENT_IN       stEnableMotionEndEvent_in;
MMC_ENABLEMOTIONENDEDEVENT_OUT      stEnableMotionEndEvent_out;
//
// Inserting the structure parameters:
stEnableMotionEndEvent_in.dummy     = 1;      //Dummy input
//
rc = MMC_EnableMotionEndedEventCmd (hConn, iAxisRef, &stEnableMotionEndEvent_in,
&stEnableMotionEndEvent_out);
if (rc != 0)
{
    HandleError();
}
```

12.20.4. MMC_GetEventsMask

Returns the 32 bit events mask for a specific connection.

```
MMC_LIB_API int MMC_GetEventsMaskCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_GETEVENTSMASK_IN* pInParam,
OUT MMC_GETEVENTSMASK_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_events_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_GETEVENTSMASK_IN** input data structure using the MMC_GetEventsMask function.

pOutParam

Points to the **MMC_GETEVENTSMASK_OUT** output structure receiving information, as a result of calling the MMC_GetEventsMask function.

Remarks

None

Scope

All

MMC_GETEVENTSMASK_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_GETEVENTSMASK_IN;
```

Parameters

dummy

Dummy input. Any +ve character value.

MMC_GETEVENTSMASK_OUT Structure

```
typedef struct{
int iEventsMask;
unsigned short usStatus;
short usErrorID;
}MMC_GETEVENTSMASK_OUT;
```

Parameters

iEventsMask

Defined according to the event IDs described in the section **12.18 Events Enumeration on page 12-1010.** +ve integer Bitwise ID values.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 12-6 describes the function block for MMC_GetEventsMask

Figure 12-6: MMC_GetEventsMask function block

12.20.4.2. Function Block Code Example

```
int rc;
MMC_GETEVENTSMASK_IN        stGetEventsMask_in;
MMC_GETEVENTSMASK_OUT       stGetEventsMask_out;
//
// Inserting the structure parameters:
stGetEventsMask_in.dummy        = 1;     //Dummy input
//
rc = MMC_GetEventsMaskCmd (hConn, &stGetEventsMask_in, &stGetEventsMask_out);
printf("Events Mask Status[%ld] ErrId[%d]\n", (long int)stGetEventsMask_out.iEventsMask,
(short)stGetEventsMask_out.usErrorID);
if (rc != 0)
{
    HandleError();
}
```

12.20.5. MMC_SetEventsMask

Sets the 32-bit events mask for a specific connection defined by the input mask parameter *iEventsMask*.

```
MMC_LIB_API int MMC_SetEventsMaskCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_SETEVENTSMASK_IN* pInParam,
OUT MMC_SETEVENTSMASK_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - Supported
Source	GMAS\includes\MMC_events_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_SETEVENTSMASK_IN** input data structure using the MMC_SetEventsMask function.

pOutParam

Points to the **MMC_SETEVENTSMASK_OUT** output structure receiving information, as a result of calling the MMC_SetEventsMask function.

Remarks

None

Scope

All

MMC_SETEVENTSMASK_IN Structure

```
typedef struct{
int iEventsMask;
}MMC_SETEVENTSMASK_IN;
```

Parameters

iEventsMask

Defined according to the event IDs described in the section **12.18 Events Enumeration on page 12-1010.** +ve integer Bitwise ID values.

MMC_SETEVENTSMASK_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_SETEVENTSMASK_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 12-7 describes the function block for MMC_SetEventsMask

Figure 12-7: MMC_SetEventsMask function block

12.20.5.2. Function Block Code Example

```
int rc;
MMC_SETEVENTSMASK_IN        stSetEventsMask_in;
MMC_SETEVENTSMASK_OUT       stSetEventsMask_out;
//
// Inserting the structure parameters:
stSetEventsMask_in.iEventsMask = 7;     //Dummy input
//
rc = MMC_SetEventsMaskCmd (hConn, &stSetEventsMask_in, &stSetEventsMask_out);
if (rc != 0)
{
    HandleError();
}
```

Chapter 13: G-MAS Personality

The G-MAS personality file is the individual character of a G-MAS stored in an XML formatted file. It is situated and named in the directory:

…../var/MMC/config/resources/Personlity.xml.

This file includes the following information:

Data recording parameters	Periodically, the GMAS updates its data recording parameters. Instead of performing modifications in the host software (i.e EAS application) to support these changes, the parameter's list which includes; size, units, and type, are detailed in this file.
Error List and resolution	When an error is returned from the G-MAS, 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 G-MAS parameters which is to be viewed in the EAS. Each parameter has the following attributes: • Number • Name • Type • Array size • Permissions (RO, save to flash, etc …) • Min / MAX / Default values

13.1. G-MAS Personality Functions

The following G-MAS Personality Functions are described:

G-MAS Personality Functions
MMC_CreatePersonalityParameters

13.1.1. MMC_CreatePersonalityParameters

Create a set of G-MAS character parameters.

```
MMC_LIB_API int MMC_CreatePersonalityParametersCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_CREATEPERSONALITYPARAMETERS_IN* pInParam,
OUT MMC_CREATEPERSONALITYPARAMETERS_OUT* pOutParam
);
```

Motion Mode	NC - ?	Distributed - ?
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_CANCELGENERALRPDO3_IN** input data structure using the MMC_CreatePersonalityParameters function.

pOutParam

Points to the **MMC_CANCELGENERALRPDO3_OUT** output structure receiving information, as a result of calling the MMC_CreatePersonalityParameters function.

Remarks

None

Scope

None

MMC_CREATEPERSONALITYPARAMETERS_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_CREATEPERSONALITYPARAMETERS_IN;
```

Parameters

ucDummy

Dummy values. Any +ve character value.

MMC_CREATEPERSONALITYPARAMETERS_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_CREATEPERSONALITYPARAMETERS_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 13-1 describes the function block for MMC_CreatePersonalityParameters

Figure 13-1: MMC_CreatePersonalityParameters function block

13.1.1.2. Function Block Code Example

Chapter 14: Data Recording

Data recording is a powerful feature of the G-MAS that allows the user to record internal controller variables, store them in local a temporary array, and upload them to a host computer using either one of the controller's communication channels.

Caution: This is an advanced API option. Care should be taken when using data recording. It should only be used when the G-MAS parameters have been set and the servo drivers functioning correctly. Only use TCP/IP communication to perform the data recording.

The G-MAS has the following data recording capabilities:

- Simultaneous recording of up to a maximum of 32 bit internal controller signals/variables
- Up to one million data recorded points. The user can select to record up to a maximum of 32 bit vectors with 31,250 sample points, or for single vectors, one million sample points, or any other combination
- Various recording vectors
- Advanced triggering options

In most situations where the variable is described as a char, short, int, float, etc., each variable can be recorded as a single vector, and therefore a maximum 32 bit variable is recorded. However, where the variable is a double, the vector memory space allocated requires two vectors, therefore only allowing a 16 bit variable to be recorded.

14.1. Triggering a Recording

The G-MAS supports advanced trigger options that further enhance the data logging capabilities of the system, and provide a powerful tool for monitoring and debugging servo applications. In general, Trigger support refers to the ability of the controller to start a data recording process, and to condition the actual execution of the data logging based on a specific event. The user can select the event source, type, and condition, as well as to perform pre-trigger data logging (logging data prior to the trigger event).

The G-MAS support numerous trigger event types, generally divided into the following groups:

Group	Explanation
Edge Type Triggers	Where a change in signal level around some threshold is detected.
Level Type Triggers	Where the signal level is detected (not necessarily a change).
Single Level or Window conditions	The trigger condition can be defined as a single level or a window (Min/Max).
Bitwise Masking of signals	The trigger condition can be defined by a user defined Bitwise mask of the requested signal.

14.2. Active Range Support

The fast upload data rate supported via the TCP/IP link, and double buffering capability, allows the host computer to display data recording in an active range, with all of the triggering options. There are four data recording types:

Type	Explanation
AUTO	The data recording is performed with no triggering, but is cyclic (the double buffering recording mechanism is used).
SINGLE	The data recording is performed with triggers, but is not cyclic. The trigger is searched for, once only, and then the recording ends.
NOTRIGGER	The data recording is performed with no triggering, and is not cyclic (The double buffering recording mechanism is not used).
NORMAL	The data recording is performed with triggering, and is cyclic (the double buffering recording mechanism is used).

14.3. Using Data Recording in the G-MAS

The GMAS supports Data Recording using the following keywords:

Keywords	Variable	Command
Begin / Stop Data Recording command		MMC_BeginRecordingCmd
Data Recording Configuration Parameters:		MMC_BEGIN_RECORDING_IN
• Select Recorded variables parameter	uiRp	
• Select Recording Length parameter	uiRl	
• Select Recording GAP parameter	uiRg	
Report Recording Status command Rest Recording Index	uiRr	MMC_RecStatusCmd
Uploading the array From/To/Buffered index	uiFrom uiTo uiBufIdx	MMC_UploadDataCmd
Triggering Options Recorder Trigger Status	uiSr	
Data Recording Array		

14.3.1. Excluding Triggers

When normal data recording without triggers is used, the Recording Status variable uiRr is automatically initialized by the Begin Recording command, with the user defined Recording Length uiRr=uiRl (number of recorded points). During the recording process it is decremented, every gap (defined by uiRg) servo samples by 1 until it reaches 0 (Rr=0). At this point, the data recording is terminated and the data itself can be uploaded.

14.3.2. Including Triggers

The ability to perform data recording with Triggers Support general depends on the capability of the controller to initiate a data recording process, and to condition its actual execution and progress bases on user-defined events and conditions. For the Pre-Triggering phase, the controller supports data logging before the trigger event occurred. The GMAS supports pre-trigger buffer length from 0% to 100%.

The Arming (using variable uiSr) state indicates that the controller is in the Pre-Trigger phase, collecting data, and filling the recording buffer, to the size of pre-trigger size defined. When using triggers, the phase starts immediately with the Start Recording command. The variable uiRr is initially set to uiRl, and is decremented until the size of the Pre-Trigger is reached. For example, if uiRl =1000, and the Pre-Trigger is 25% - i.e. 250 points, uiRr will be decremented during the Pre-Trigger state to 750.

The condition whereby the system is waiting for an Inverse Trigger state is present only in Edge Trigger modes, and indicates that the controller is waiting for the opposite trigger condition, to operate the trigger condition itself. This is required for Edge or Change Detection. At this phase, data logging is continuously performed in the internal data recording buffers, but the actual recording is paused. The Recording Status variable uiRr is unchanged.

The delay for a Trigger Condition state is present in all Trigger modes, to indicate that the controller is waiting for the trigger event condition. At this phase, data logging is continuously performed in the internal data recording buffers, but the actual recording is paused. Similarly, the Recording Status variable uiRr is unchanged.

When the Triggered and Recording state is invoked, the trigger event is detected, and normal data recoding continues. The End of Recording process always completes when the Recording Status Variable reach Zero value uiRr =0.

When using normal data recording without triggers, the Start of Recording (Begin Recording Command) is referred to the 0 time-base in the output data buffer shown on the graphic plots. When operating with triggers, the 0 time-base will depend on the trigger Event condition. The actual Start of Recording time is no longer relevant, as the trigger event condition can occur long after the Start Recording command was issued. In this case, the 0 time-base of the Output Buffers refers to the point that is the Pre-Trigger time before the Trigger Event. For example, if a 1 second recording process is initiated, with a pre-trigger of 50%, the Pre-Trigger point is always (by definition) at 0.5 second after start.

14.4. Recording Definitions and Parameters

The following is a list of recording enumerator definitions and parameters.

14.4.1. Recording Data Signals Bitmask Definitions

The Recording Data Signal Bitmask variable uiRc, and ulRc create a mask using the following memory buffers to store the input parameters.

Bit Mask Parameter	Memory used
MC_REC_TRIGGER_TYPE_MASK	0x0000ffff
MC_REC_TRIGGER_PARAM_MASK	0xffff0000
MC_REC_TRIGGERG_STATE_MASK	0x00ff
MC_REC_BUF_STATE_MASK	0xff00
MC_SCOPE_BITS_NONE_BUF_READY	0x000
MC_SCOPE_BITS_BUFFER1_READY	0x100
MC_SCOPE_BITS_BUFFER2_READY	0x200
MC_NORMAL_TRIGGER	0x10000
MC_AUTO_TRIGGER	0x20000
MC_SINGLE_TRIGGER	0x30000

14.4.2. Recording Signal Parameters

The following parameters are recording signal variables and their IDs.

Recording Signal Variable	ID
NC_REC_DESIRED_POS_LOW_PARAM	0
NC_REC_DESIRED_POS_HIGH_PARAM	1
NC_REC_DESIRED_VEL_LOW_PARAM	2
NC_REC_DESIRED_VEL_HIGH_PARAM	3
NC_REC_GROUP_VEL_LOW_PARAM	4
NC_REC_GROUP_VEL_HIGH_PARAM	5
NC_REC_GROUP_AC_LOW_PARAM	6
NC_REC_GROUP_AC_HIGH_PARAM	7
NC_REC_GROUP_DC_LOW_PARAM	8
NC_REC_GROUP_DC_HIGH_PARAM	9

Recording Signal Variable	ID
NC_REC_GROUP_AC_DC_LOW_PARAM	10
NC_REC_GROUP_AC_DC_HIGH_PARAM	11
NC_REC_GROUP_JERK_LOW_PARAM	12
NC_REC_GROUP_JERK_HIGH_PARAM	13
NC_REC_SMOOTH_FACTOR_AC_LOW_PARAM	14
NC_REC_SMOOTH_FACTOR_AC_HIGH_PARAM	15
NC_REC_SMOOTH_FACTOR_DC_LOW_PARAM	16
NC_REC_SMOOTH_FACTOR_DC_HIGH_PARAM	17
NC_REC_POS_INCR_LOW_PARAM	18
NC_REC_POS_INCR_HIGH_PARAM	19
NC_REC_CYCLE_CNT_PARAM	20
NC_REC_TARGET_POS_PARAM	21
NC_REC_TARGET_VEL_PARAM	22
NC_REC_F_POS_PARAM	23
NC_REC_F_VEL_PARAM	24
NC_REC_ACTUAL_POS_PARAM	25
NC_REC_ACTUAL_VEL_PARAM	26
NC_REC_AXIS_STATUS_PARAM	27
NC_REC_MAX_NUM_PARAM	28
NC_REC_ACTUAL_POS_PARAM	30
NC_REC_ACTUAL_VEL_PARAM	31
NC_REC_AXIS_STATUS_PARAM	32
NC_REC_ACTUAL_TORQUE_PARAM	33
NC_REC_I_USER_1_PARAM	34
NC_REC_I_USER_AUX_1_PARAM	35
NC_REC_F_USER_1_PARAM	36
NC_REC_F_USER_AUX_1_PARAM	37
NC_REC_I_USER_2_PARAM	38
NC_REC_I_USER_AUX_2_PARAM	39
NC_REC_F_USER_2_PARAM	40
NC_REC_F_USER_AUX_2_PARAM	41

Recording Signal Variable	ID
NC_REC_I_USER_3_PARAM	42
NC_REC_I_USER_AUX_3_PARAM	43
NC_REC_F_USER_3_PARAM	44
NC_REC_F_USER_AUX_3_PARAM	45
NC_REC_I_USER_4_PARAM	46
NC_REC_I_USER_AUX_4_PARAM	47
NC_REC_F_USER_4_PARAM	48
NC_REC_F_USER_AUX_4_PARAM	49
NC_REC_POS_FOLLOWING_ERR_PARAM	50
NC_REC_DIGITAL_INPUTS_PARAM	51
NC_REC_DIGITAL_OUTPUTS_PARAM	52
NC_REC_TRACKING_ERROR_LOW_PARAM	53
NC_REC_TRACKING_ERROR_HIGH_PARAM	54
NC_REC_ERROR_CORRECTION_POS_PARAM	55
NC_REC_ACTUAL_HW_POSITION_PARAM	56
NC_REC_CONTROL_WORD_PARAM	57
NC_REC_STATUS_WORD_PARAM	58
NC_REC_MOTION_MODE_PARAM	59
NC_REC_DI_LOW_PARAM	60
NC_REC_DI_HIGH_PARAM	61
NC_REC_DO_LOW_PARAM	62
NC_REC_DO_HIGH_PARAM	63
NC_REC_AXIS_COMM_ERROR_PARAM	64
NC_REC_AXIS_LAST_EMCY_CODE_PARAM	65
NC_STATUS_REGISTER	66
NC_MCS_LIMIT_REGISTER	67
NC_REC_DESIRED_MCS_X_POS_LOW_PARAM	68
NC_REC_DESIRED_MCS_X_POS_HIGH_PARAM	69
NC_REC_DESIRED_MCS_Y_POS_LOW_PARAM	70
NC_REC_DESIRED_MCS_Y_POS_HIGH_PARAM	71
NC_REC_DESIRED_MCS_Z_POS_LOW_PARAM	72
NC_REC_DESIRED_MCS_Z_POS_HIGH_PARAM	73
NC_REC_DESIRED_MCS_U_POS_LOW_PARAM	74
NC_REC_DESIRED_MCS_U_POS_HIGH_PARAM	75

Recording Signal Variable	ID
NC_REC_DESIRED_MCS_V_POS_LOW_PARAM	76
NC_REC_DESIRED_MCS_V_POS_HIGH_PARAM	77
NC_REC_DESIRED_MCS_W_POS_LOW_PARAM	78
NC_REC_DESIRED_MCS_W_POS_HIGH_PARAM	79
NC_REC_DESIRED_MCS_N1_POS_LOW_PARAM	80
NC_REC_DESIRED_MCS_N1_POS_HIGH_PARAM	81
NC_REC_DESIRED_MCS_N2_POS_LOW_PARAM	82
NC_REC_DESIRED_MCS_N2_POS_HIGH_PARAM	83
NC_REC_DESIRED_MCS_N3_POS_LOW_PARAM	84
NC_REC_DESIRED_MCS_N3_POS_HIGH_PARAM	85
NC_REC_DESIRED_MCS_N4_POS_LOW_PARAM	86
NC_REC_DESIRED_MCS_N4_POS_HIGH_PARAM	87
NC_REC_DESIRED_MCS_N5_POS_LOW_PARAM	88
NC_REC_DESIRED_MCS_N5_POS_HIGH_PARAM	89
NC_REC_DESIRED_MCS_N6_POS_LOW_PARAM	90
NC_REC_DESIRED_MCS_N6_POS_HIGH_PARAM	91
NC_REC_DESIRED_MCS_N7_POS_LOW_PARAM	92
NC_REC_DESIRED_MCS_N7_POS_HIGH_PARAM	93
NC_REC_DESIRED_MCS_N8_POS_LOW_PARAM	94
NC_REC_DESIRED_MCS_N8_POS_HIGH_PARAM	95
NC_REC_DESIRED_MCS_N9_POS_LOW_PARAM	96
NC_REC_DESIRED_MCS_N9_POS_HIGH_PARAM	97
NC_REC_DESIRED_MCS_S_POS_LOW_PARAM	98
NC_REC_DESIRED_MCS_S_POS_HIGH_PARAM	99
NC_REC_DESIRED_MCS_X_VEL_LOW_PARAM	100
NC_REC_DESIRED_MCS_X_VEL_HIGH_PARAM	101
NC_REC_DESIRED_MCS_Y_VEL_LOW_PARAM	102
NC_REC_DESIRED_MCS_Y_VEL_HIGH_PARAM	103
NC_REC_DESIRED_MCS_Z_VEL_LOW_PARAM	104
NC_REC_DESIRED_MCS_Z_VEL_HIGH_PARAM	105
NC_REC_DESIRED_MCS_U_VEL_LOW_PARAM	106
NC_REC_DESIRED_MCS_U_VEL_HIGH_PARAM	107
NC_REC_DESIRED_MCS_V_VEL_LOW_PARAM	108
NC_REC_DESIRED_MCS_V_VEL_HIGH_PARAM	109
NC_REC_DESIRED_MCS_W_VEL_LOW_PARAM	110

Recording Signal Variable	ID
NC_REC_DESIRED_MCS_W_VEL_HIGH_PARAM	111
NC_REC_DESIRED_MCS_N1_VEL_LOW_PARAM	112
NC_REC_DESIRED_MCS_N1_VEL_HIGH_PARAM	113
NC_REC_DESIRED_MCS_N2_VEL_LOW_PARAM	114
NC_REC_DESIRED_MCS_N2_VEL_HIGH_PARAM	115
NC_REC_DESIRED_MCS_N3_VEL_LOW_PARAM	116
NC_REC_DESIRED_MCS_N3_VEL_HIGH_PARAM	117
NC_REC_DESIRED_MCS_N4_VEL_LOW_PARAM	118
NC_REC_DESIRED_MCS_N4_VEL_HIGH_PARAM	119
NC_REC_DESIRED_MCS_N5_VEL_LOW_PARAM	120
NC_REC_DESIRED_MCS_N5_VEL_HIGH_PARAM	121
NC_REC_DESIRED_MCS_N6_VEL_LOW_PARAM	122
NC_REC_DESIRED_MCS_N6_VEL_HIGH_PARAM	123
NC_REC_DESIRED_MCS_N7_VEL_LOW_PARAM	124
NC_REC_DESIRED_MCS_N7_VEL_HIGH_PARAM	125
NC_REC_DESIRED_MCS_N8_VEL_LOW_PARAM	126
NC_REC_DESIRED_MCS_N8_VEL_HIGH_PARAM	127
NC_REC_DESIRED_MCS_N9_VEL_LOW_PARAM	128
NC_REC_DESIRED_MCS_N9_VEL_HIGH_PARAM	129
NC_REC_DESIRED_MCS_S_VEL_LOW_PARAM	130
NC_REC_DESIRED_MCS_S_VEL_HIGH_PARAM	131
NC_REC_DESIRED_MCS_X_AC_DC_LOW_PARAM	132
NC_REC_DESIRED_MCS_X_AC_DC_HIGH_PARAM	133
NC_REC_DESIRED_MCS_Y_AC_DC_LOW_PARAM	134
NC_REC_DESIRED_MCS_Y_AC_DC_HIGH_PARAM	135
NC_REC_DESIRED_MCS_Z_AC_DC_LOW_PARAM	136
NC_REC_DESIRED_MCS_Z_AC_DC_HIGH_PARAM	137
NC_REC_DESIRED_MCS_U_AC_DC_LOW_PARAM	138
NC_REC_DESIRED_MCS_U_AC_DC_HIGH_PARAM	139
NC_REC_DESIRED_MCS_V_AC_DC_LOW_PARAM	140
NC_REC_DESIRED_MCS_V_AC_DC_HIGH_PARAM	141
NC_REC_DESIRED_MCS_W_AC_DC_LOW_PARAM	142
NC_REC_DESIRED_MCS_W_AC_DC_HIGH_PARAM	143
NC_REC_DESIRED_MCS_N1_AC_DC_LOW_PARAM	144
NC_REC_DESIRED_MCS_N1_AC_DC_HIGH_PARAM	145

Recording Signal Variable	ID
NC_REC_DESIRED_MCS_N2_AC_DC_LOW_PARAM	146
NC_REC_DESIRED_MCS_N2_AC_DC_HIGH_PARAM	147
NC_REC_DESIRED_MCS_N3_AC_DC_LOW_PARAM	148
NC_REC_DESIRED_MCS_N3_AC_DC_HIGH_PARAM	149
NC_REC_DESIRED_MCS_N4_AC_DC_LOW_PARAM	150
NC_REC_DESIRED_MCS_N4_AC_DC_HIGH_PARAM	151
NC_REC_DESIRED_MCS_N5_AC_DC_LOW_PARAM	152
NC_REC_DESIRED_MCS_N5_AC_DC_HIGH_PARAM	153
NC_REC_DESIRED_MCS_N6_AC_DC_LOW_PARAM	154
NC_REC_DESIRED_MCS_N6_AC_DC_HIGH_PARAM	155
NC_REC_DESIRED_MCS_N7_AC_DC_LOW_PARAM	156
NC_REC_DESIRED_MCS_N7_AC_DC_HIGH_PARAM	157
NC_REC_DESIRED_MCS_N8_AC_DC_LOW_PARAM	158
NC_REC_DESIRED_MCS_N8_AC_DC_HIGH_PARAM	159
NC_REC_DESIRED_MCS_N9_AC_DC_LOW_PARAM	160
NC_REC_DESIRED_MCS_N9_AC_DC_HIGH_PARAM	161
NC_REC_DESIRED_MCS_S_AC_DC_LOW_PARAM	162
NC_REC_DESIRED_MCS_S_AC_DC_HIGH_PARAM	163
NC_REC_END_MOTION_REASON_PARAM	164
NC_REC_ANALOG_INPUT_PARAM	165
NC_REC_MAX_NUM_PARAM	166

14.4.3. Trigger Modes

The parameter uiRp controls the trigger value and type. These trigger values consist of high and low bytes. The uiRp[1] higher 2 bytes control the axis or node ID, wheras the uiRp[3] low 2 bytes may control the motion. Under these conditions uiRp[1] low 2 bytes should be 0. The uiRp[3] high 2 bytes control the trigger type which may be of three values as detailed in the section **14.4.1** above:

Bit Mask Parameter	Memory used
MC_NORMAL_TRIGGER	0x10000
MC_AUTO_TRIGGER	0x20000
MC_SINGLE_TRIGGER	0x30000

The following Trigger modes and their IDs are used in the Recording Data functions.

Parameter	ID	Explanation
TG_RECORDING_TRIGGER_TYPE_NO_TRIGGER	0	Edge : Rising (Positive Slope Over: TRIGVAL >= Level#1)
TG_RECORDING_TRIGGER_TYPE_EDGE_Rise	1	Edge : Rising (Positive Slope Over: TRIGVAL >= Level#1)
TG_RECORDING_TRIGGER_TYPE_EDGE_Fall	2	Edge : Falling (Negative Slope over: TRIGVAL <= Level#1)
TG_RECORDING_TRIGGER_TYPE_EDGE_WindowIn	3	Edge : Window In (Into the Window defined by: Level#2 <= TRIGVAL <= LEVEL#1)
TG_RECORDING_TRIGGER_TYPE_EDGE_WindowOut	4	Edge : Window Out (Out Of the Window defined by: Level#2 <= TRIGVAL <= LEVEL#1)
TG_RECORDING_TRIGGER_TYPE_LEVEL_GE	5	Level : >= (GreaterEqual The: TRIGVAL >= Level#1)
TG_RECORDING_TRIGGER_TYPE_LEVEL_SE	6	Level : <= (SmallerEqual Then: TRIGVAL <= Level#1)
TG_RECORDING_TRIGGER_TYPE_LEVEL_WindowInside	7	Level : Inside Window (Inside of Window defined by: Level#2 <= TRIGVAL <= LEVEL#1)
TG_RECORDING_TRIGGER_TYPE_LEVEL_WindowOutside	8	Level : outside Window (Outside of Window defined by: Level#2 <= TRIGVAL <= LEVEL#1)
TG_RECORDING_TRIGGER_TYPE_EDGE_Rise_Mask	9	Rising-Edge + MASK (Positive Slope Over: (TRIGVAL & MASK) == MASK)
TG_RECORDING_TRIGGER_TYPE_EDGE_Fall_Mask	10	Falling-Edge + MASK (Negative Slope Over: (TRIGVAL & MASK) != MASK)

TG_RECORDING_TRIGGER_TYPE_LEVEL_GE_Mask	11	Grater-Equal + Mask (Equal TO: (TRIGVAL & MASK) == MASK)
TG_RECORDING_TRIGGER_TYPE_LEVEL_SE_Mask	12	Smaller-Equal + Mask (Not Equal TO: (TRIGVAL & MASK) != MASK)
TG_RECORDING_TRIGGER_TYPE_BEGIN_MOTION_Mask	13	Begin Motion

14.5. Data Recording Functions

14.5.1. MMC_BeginRecording

Starts the recording of internal controller variables data from the G-MAS server.

```
MMC_LIB_API int MMC_BeginRecordingCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_BEGIN_RECORDING_IN* pInParam,
OUT MMC_BEGIN_RECORDING_OUT* pOutParam
);
```

Motion Mode	NC – N/A	Distributed – N/A
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_BEGIN_RECORDING_IN** input data structure using the MMC_BeginRecording function.

pOutParam

Points to the **MMC_BEGIN_RECORDING_OUT** output structure receiving information, as a result of calling the MMC_BeginRecording function.

Remarks

None

Scope

All

MMC_BEGIN_RECORDING_IN Structure

```
typedef struct{
unsigned long uiRg;
unsigned long uiRl;
unsigned long uiRc;
unsigned long uiRv[NC_MAX_REC_SIGNALS_NUM];
unsigned long uiRp[NC_MAX_REC_PARAMS_NUM];
}MMC_BEGIN_RECORDING_IN;
```

Parameters

uiRg

Recording Data Gap which specifies the sampling rate of the recorder. Any +ve integer value

uiRl

Recording Data Length with a buffer size (default size 4MB). Any +ve integer value.

uiRc

Recording Data Signals Bit mask according to the definitions described in section **14.4.1 Recording Data Signals Bitmask Definitions**. Defines which of mapped signals should be recorded with up to 32 different synchronized signals. Any +ve integer value.

uiRv

uiRv is the recording Signals ID mapping enumerator which maps the IDs of the recordable signals to logical IDs that the recorder can reference. It is a 32 bit mask assembled from the AxisNumber and the Signal parameter. The upper 16 bits are the axis reference, and the lower 16 bits the signal parameter, e.g. 0x00020015, where the 0002 refers to axis 02, and the 0015 refers to the signal ID 20 (in Hex). Refer to the ID definitions described in section **14.4.2 Recording Signal Parameters**.

[NC_MAX_REC_SIGNALS_NUM] is an array value of between [1….22] and *uiRv* can have any +ve integer value.

uiRp

uiRp is the ID integer value of the Recording Parameters. Refer to the section **14.4.3 Trigger Modes** for further details. The recorder parameters defines which event will trigger the recorder, and the trigger position, according to the following definitions:

Recording Parameters - RP[N]	ID	Definition
TG_RECORDING_SPARE	0	Spare
TG_RECORDING_TRIGGER_VALUE	1	Defines which of mapped signals should be recorded, but only 1 bit may be non-zero. The trigger variable does not need to be one of the recorded variables.
TG_RECORDING_PRE_TRIGGER_LENGTH	2	The percentage of the recorded signal taken before the trigger event. (recorder

		trigger delay*)
TG_RECORDING_TRIGGER_TYPE	3	
TG_RECORDING_TRIGGER_LEVEL_1	4	Level for positive slope trigger, or high side for window trigger.
TG_RECORDING_TRIGGER_LEVEL_2	5	Level for negative slope trigger, or low side for window trigger.
TG_RECORDING_TRIGGER_POLARITY	6	Logic for bit field trigger- 0 positive logic, 1 – negative
TG_RECORDING_TRIGGER_IN_MASK	7	Mask for bit field trigger

[NC_MAX_REC_PARAMS_NUM] is an array value of [1….8].

MMC_BEGIN_RECORDING_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_BEGIN_RECORDING_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 14-1 describes the function block for MMC_BeginRecording

Figure 14-1: MMC_BeginRecording function block

14.5.1.2. Function Block Code Example

```
int rc;
MMC_BEGIN_RECORDING_IN     stBeginRecording_in;
MMC_BEGIN_RECORDING_OUT    stBeginRecording_out;
//
// Inserting the structure parameters:
stBeginRecording_in.uiRg        = 1000;       // Recording Data Gap
stBeginRecording_in.uiRl        = 10;         // Recording Data Length
stBeginRecording_in.uiRc        = 0x10000;    // Parameter array index
stBeginRecording_in.uiRv[1]     = 0x0020015;           // Recording axis number and the Signals
ID mapping
stBeginRecording_in.uiRv[2]     = 0x0030015;
stBeginRecording_in.uiRv[3]     = 0x0040015;
stBeginRecording_in.uiRp[1]     = 1; //Recorder parameters defines which event will trigger the
recorder
stBeginRecording_in.uiRp[2]     = 2;
stBeginRecording_in.uiRp[3]     = 3;
//
rc = MMC_BeginRecordingCmd (hConn, &stBeginRecording_in, &stBeginRecording_out);
if (rc != 0)
{
    HandleError() ;
}
```

14.5.1.3. Implementation Example

14.5.2. MMC_StopRecording

Halts recording of the G-MAS server data.

```
MMC_LIB_API int MMC_StopRecordingCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_STOP_RECORDING_IN* pInParam,
OUT MMC_STOP_RECORDING_OUT* pOutParam
);
```

Motion Mode	NC – N/A	Distributed – N/A
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_STOP_RECORDING_IN** input data structure using the MMC_StopRecording function.

pOutParam

Points to the **MMC_STOP_RECORDING_OUT** output structure receiving information, as a result of calling the MMC_StopRecording function.

Remarks

None

Scope

All

MMC_STOP_RECORDING_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_STOP_RECORDING_IN;
```

Parameters

dummy

Dummy values. Any +ve character value.

MMC_STOP_RECORDING_OUT Structure

```
typedef struct{
unsigned short usStatus;
short usErrorID;
}MMC_STOP_RECORDING_OUT;
```

Parameters

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 14-2 describes the function block for MMC_StopRecording

Figure 14-2: MMC_StopRecording function block

14.5.2.2. Function Block Code Example

```
int rc;
MMC_STOP_RECORDING_IN       stStopRecording_in;
MMC_STOP_RECORDING_OUT      stStopRecording_out;
//
// Inserting the structure parameters:
stStopRecording_in.dummy    = 1;      // dummy data
//
rc = MMC_StopRecordingCmd (hConn, &stStopRecording_in, &stStopRecording_out);
if (rc != 0)
{
    HandleError() ;
}
```

14.5.2.3. Implementation Example

14.5.3. MMC_UploadData

Uploads recording data to the G-MAS.

```
MMC_LIB_API int MMC_UploadDataCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_UPLOAD_DATA_IN* pInParam,
OUT MMC_UPLOAD_DATA_OUT* pOutParam
);
```

Motion Mode	NC – N/A	Distributed – N/A
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_UPLOAD_DATA_IN** input data structure using the MMC_UploadData function.

pOutParam

Points to the **MMC_UPLOAD_DATA_OUT** output structure receiving information, as a result of calling the MMC_UploadData function.

Remarks

None

Scope

All

MMC_UPLOAD_DATA_IN Structure

```
typedef struct{
unsigned int uiFrom;
unsigned int uiTo;
unsigned int uiBufIdx;
}MMC_UPLOAD_DATA_IN;
```

Parameters

uiFrom

Upload from index. Any +ve integer value.

uiTo

Upload to index. Any +ve integer value.

uiBufIdx

Buffer Index. Any +ve integer value.

MMC_UPLOAD_DATA_OUT Structure

```
typedef struct{
long ulUpdatData[NC_MAX_LONG];
unsigned short usStatus;
short usErrorID;
}MMC_UPLOAD_DATA_OUT;
```

Parameters

ulUpdatData

Update data status. Any +ve or –ve integer value, with [NC_MAX_LONG] having array values [1 - ??]

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Figure 14-3 describes the function block for MMC_UploadData

Figure 14-3: MMC_UploadData function block

14.5.3.2. Function Block Code Example

```
int rc;
MMC_UPLOAD_DATA_IN          stUploadData_in;
MMC_UPLOAD_DATA_OUT         stUploadData_out;
//
// Inserting the structure parameters:
stUploadData_in.uiFrom      = 1;      // Upload from index
stUploadData_in.uiTo        = 100;    // Upload to index
stUploadData_in.uiBufIdx    = 1000;   // Buffer Index
//
rc = MMC_UploadDataCmd (hConn, &stUploadData_in, &stUploadData_out);
if (rc != 0)
{
    HandleError() ;
}
```

14.5.3.3. Implementation Example

14.5.4. MMC_RecStatus

Requests the status of the recording.

```
MMC_LIB_API int MMC_RecStatusCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_REC_STATUS_IN* pInParam,
OUT MMC_REC_STATUS_OUT* pOutParam
);
```

Motion Mode	NC – N/A	Distributed – N/A
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pInParam

Points to the **MMC_REC_STATUS_IN** input data structure using the MMC_RecStatus function.

pOutParam

Points to the **MMC_REC_STATUS_OUT** output structure receiving information, as a result of calling the MMC_RecStatus function.

Remarks

None

Scope

All

MMC_REC_STATUS_IN Structure

```
typedef struct{
unsigned char dummy;
}MMC_REC_STATUS_IN;
```

Parameters

dummy

Dummy values. Any +ve character value.

MMC_REC_STATUS_OUT Structure

```
typedef struct{
unsigned long uiRr;
unsigned long uiSr;
unsigned short usStatus;
short usErrorID;
}MMC_REC_STATUS_OUT;
```

Parameters

uiRr

Rest Recording Index. Reads back recorder status. Any +ve integer value

uiSr

Recorder Trigger Status. Status register, which indicates the status of the recorder; idle, armed, triggered and recording, or ready with data, where the lower 8 bits display the following options to be entered:

0 – Arming

1 – Waiting Opposite trigger

2 – Waiting Trigger

3 – Trigger Detected

4 – No Trigger

The following 8 bits (bitwise) display the following options to be entered:

0 – No Buffer Ready

1 – Buffer 1 ready

2 – Buffer 2 ready

3 – Both Buffers ready

Values accepted are any +ve integer value.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 14-4 describes the function block for MMC_RecStatus

Figure 14-4: MMC_RecStatus function block

14.5.4.2. Function Block Code Example

```
int rc;
MMC_REC_STATUS_IN      stRecStatus_in;
MMC_REC_STATUS_OUT     stRecStatus_out;
//
// Inserting the structure parameters:
stRecStatus_in.dummy   = 1;      // dummy data
//
rc = MMC_RecStatusCmd (hConn, &stRecStatus_in, &stRecStatus_out);
if (rc != 0)
{
    HandleError() ;
}
```

14.5.4.3. Implementation Example

14.5.5. MMC_UploadDataHeader

Recorder upload data header.

```
MMC_LIB_API int MMC_UploadDataHeaderCmd(
IN MMC_CONNECT_HNDL hConn,
OUT NC_UPLOAD_REC_HEADER_STRUCT* pOutParam
);
```

Motion Mode	NC – N/A	Distributed – N/A
Source	GMAS\includes\MMC_general_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pOutParam

[OUT] Points to the **MMC_POSITIONPROFILE_OUT** output structure receiving information, as a result of calling the MMC_UploadDataHeader function.

Remarks

None

Scope

All

NC_UPLOAD_REC_HEADER_STRUCT Output Structure

```
typedef struct {
unsigned long ulDummy;
unsigned long ulRc;
unsigned long ulRg;
unsigned long ulRl;
NC_REC_RV_STRUCT usRv[NC_MAX_REC_SIGNALS_NUM];
unsigned long ulRp[NC_MAX_REC_PARAMS_NUM];
unsigned long ulTi;
unsigned long ulTs;
unsigned long ulSpare[3];
unsigned short usStatus;
short usErrorID;
unsigned char dummy [952];
}NC_UPLOAD_REC_HEADER_STRUCT;
```

Parameters

ulDummy

Dummy data. To align and upload data field in a common message. Any +ve character value.

ulRc

Recording data signals bit mask according to the definitions described in section **14.4.1 Recording Data Signals Bitmask Definitions**. Defines which of mapped signals should be recorded with up to 32 different synchronized signals. Any +ve integer value.

ulRg

Recording data gap, which specifies the sampling rate of the recorder. Any +ve integer value.

ulRl

Recording data length with a buffer size (default size 4MB). Any +ve integer value.

usRv

usRv is the recording Signals ID mapping enumerator which maps the IDs of the recordable signals to logical IDs that the recorder can reference. Refer to the ID definitions described in section **14.4.2 Recording Signal Parameters** and **14.4.3 Trigger Modes.**

[NC_MAX_REC_SIGNALS_NUM] is an array value of between [1….32] and *usRv* can have any +ve integer value.

NC_REC_RV_STRUCT is the recorder signal value structure with the following parameters:

ulValue

Signal value reference handle of the axis. Any +ve integer value.

ulType

Signal value type. The Enumerator ID has the following variable values, which describe the enumerator NC_RV_TYPE_ENUM, and are the recorder supported data types.

Recorder Supported Data Types	ID
NC_UCHAR_TYPE	0
NC_CHAR_TYPE	1
NC_USHORT_TYPE	2
NC_SHORT_TYPE	3
NC_UINT_TYPE	4
NC_INT_TYPE	5
NC_ULONG_TYPE	6
NC_LONG_TYPE	7
NC_FLOAT_TYPE	8
NC_DOUBLE_L_TYPE	9
NC_DOUBLE_H_TYPE	10

ulFactor

Signal value multiple factor of the cycle time. Any positive float value

ulRp

ulRp is the ID integer value of the Recording Parameters. Refer to the section **14.4.3 Trigger Modes** for further details. The recorder parameters defines which event will trigger the recorder, and the trigger position, according to the following definitions:

Recording Parameters - RP[N]	ID	Definition
TG_RECORDING_SPARE	0	Spare
TG_RECORDING_TRIGGER_VALUE	1	Defines which of mapped signals should be recorded, but only 1 bit may be non-zero. The trigger variable does not need to be one of the recorded variables.
TG_RECORDING_PRE_TRIGGER_LENGTH	2	The percentage of the recorded signal taken before the trigger event. (recorder trigger delay*)
TG_RECORDING_TRIGGER_TYPE	3	
TG_RECORDING_TRIGGER_LEVEL_1	4	Level for positive slope trigger, or high side for window trigger.
TG_RECORDING_TRIGGER_LEVEL_2	5	Level for negative slope trigger, or low side for window trigger.

TG_RECORDING_TRIGGER_POLARITY	6	Logic for bit field trigger- 0 positive logic, 1 – negative
TG_RECORDING_TRIGGER_IN_MASK	7	Mask for bit field trigger

[NC_MAX_REC_PARAMS_NUM] is an array value of [1….8].

ulTi

Trigger Index. Any +ve integer value.

ulTs

Recorder Update Time. Sampling time, the basic resolution of recorder, which is the basic time of the NC process. Any +ve integer value.

ulSpare[3]

Spare. Any +ve integer value with a maximum of 3 integers.

dummy [952]

Dummy data. Any +ve character value to a maximum of 952 characters.

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417 - 6-453**.

Figure 14-5 describes the function block for MMC_UploadDataHeader

Figure 14-5: MMC_UploadDataHeader function block

14.5.5.2. Function Block Code Example

```
int rc;
NC_UPLOAD_REC_HEADER_STRUCT     stUploadRecHeadStr;
//NC_REC_RV_STRUCT (ulValue, ulType, ulFactor);
//
//
rc = MMC_UploadDataHeaderCmd (hConn, &stUploadRecHeadStr);
if (rc != 0)
{
    HandleError() ;
}
```

14.5.5.3. Implementation Example

Chapter 15: Using Function Blocks in C++

15.1. Programming in C++

C++ began as enhancements to C, first adding classes, then virtual functions, operator overloading, multiple inheritance, templates, and exception handling among other features. The class has both an interface and a structure. The interface describes how to interact with the class and its instances using methods, while the structure describes how the data is partitioned into attributes within an instance. A class may also have a representation (metaobject) at run time, which provides run time support for manipulating the class-related metadata.

While Elmo has created a series of sophisticated function blocks in C, in C++ these functions are wrapped, thereby dividing the C functions into better-structured categories (classes), where these classes have methods (similar to the C functions) and parameters.

C++ has the following advantages:

- Overloading functions. User can call for example; MoveAbsolute(pos) and also MoveAbsolute(pos,vel)
- Try-catch error handling mechanisms. Instead of using the 'if' condition continuously, per function, there can be a single error handler per function, in place of numerous 'ifs' that perform the same error handling.
- Eclipse Drop-Down function list. The relevant function can be chosen from the dropdown list.
- It is not necessary to learn the parameters of each function. Once the default parameters are set, only position, and velocity, for instance, need be updated. However, it will be necessary to copy parameters, as function blocks do not all have same values.

Each Class has a function using the name *SetDefaultValues*, which receives a pointer to a structure, for example, with all motion values of the single axis. After calling this function, **all** motion parameters are copied to a local buffer in the class. After calling, the overloaded functions may be called.

15.1.1. CMMCException

This return in all C++ wrapper functions is based on this CMMCException, which normally returns 0 on success, and throws an exception on error.

Figure 15-1 Fields and methods of the CMMCException class

When an error is encountered, the following information is returned:

Function Name	Structure name	
Axis reference	Error ID	Status of the axis

15.2. The CMMCAxis class

This class is the main header for the single and multiple axes wrapper functions as described in the diagram in **Figure 15-2**. The diagram displays the heirarchial structure of the main classes and their type definitions associated with CMMCAxis.Each class consist of an *.h file containing functions which correspond to similar functions in C, and are referenced with their similar explanations and details. However, this is appropriate only to Public functions. Private functions are internal functions which are transparent to the user.

Figure 15-2 CMMCAxis class diagram

15.2.1. CMMCDS406

The purpose of encoders is to detect positions of any kind of machine tools. Encoders detect positions and transmit the position values or provide speed, acceleration, and jerk values, across the CANopen network. The encoder may receive configuration information via SDO, and in the NMT state operation, the position value may be transmitted using synchronous PDO. Additionally, the encoders may transmit a PDO asynchronously, scheduled by the elapsing of the event timer.

The CANopen device profile defines two encoder classes, a standard device class 1 (C1) and an extended device class 2 (C2). The standard device C1 specifies basic functions provided by each device. The C2 extended device provides a variety of features with mandatory and optional functions. The mandatory functions of both, C1 and C2, are necessary to ensure non-manufacturer specific operations of a device.

By defining mandatory device characteristics in C1, the operation of the basic network and encoder is ensured. By defining extended C2, a degree of defined flexibility may be built-in. By leaving 'hooks' for optional and manufacturer-specific functions, the device developer is not constrained to an out-of-date standard.

The CiA DS406 device profile for encoders specifies the CANopen interface of absolute linear and rotary encoders. Bseides position and velocity ouput, the profile describes also acceleration and jerk outputs, and specifies several configuration parameters, e.g. the code sequence (complement) that determines the counting direction, in which the output code increases or decreases. The resolution parameter is used to configure a given number of steps for each revolution. The profile specification covers complete cam functionality with hysteresis, and it is possible to describe multi-sensor modules implemented in a single CANopen encoder device.

The encoder profile specfies the following operation modes:

Mode	Profile
Event-timer	Current position value is sampled and transmitted periodically.
Synchronous	Current position is sampled and transmitted after the reception of the Sync message.

The remote mode based on remotely requested PDOs is not recommended due to several general problems that occur when CAN remote frames are used.

15.2.2. DisableMotionEndedEvent

This function disables the event of Ended Motion. The same callbacks are called as in the Singe Axis, however, the group axis reference is called.

```
void DisableMotionEndedEvent(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

DisableMotionEndedEvent

Refer to the section **12.20.2 MMC_DisableMotionEndedEvent on page 12-1022** for details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.2.3. EnableMotionEndedEvent

This function enables the event of Ended Motion. The same callbacks are called as in the Singe Axis, however, the group axis reference is called.

```
void EnableMotionEndedEvent(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

EnableMotionEndedEvent

Refer to the section **12.20.3 MMC_EnableMotionEndedEvent on page 12-1025** for details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.2.4. SetDefaultManufacturerParameters

This function restores the axis\group parameters to their original factory defaults values.

```
void SetDefaultManufacturerParameters(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

SetDefaultManufacturerParameters

Refer to the section **10.3.37 MMC_SetDefaultParametersGlobal on page 10-641** for details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.2.5. GetFbDepth

Refer to the section **5.7.2 MMC_GetFBDepth on page 5-148** for details of the description, scope, and motion mode.

```
unsigned int GetFbDepth(
const unsigned int uiHndl
)throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

GetFbDepth

Refer to the section **5.7.2 MMC_GetFBDepth on page 5-148** for details of the function.

uiHndl

Returned function block handle. Integer with any +ve value.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.2.6. GetAxisByName

Refer to the section **10.3.17 MMC_GetAxisByName on page 10-584** for details of the description, scope, and motion mode.

```
int GetAxisByName(
const char* cName
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

cName

Tag/assembly name as declared in XML configuration file.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.2.7. GetGroupAxisByName

Refer to the section **10.3.17 MMC_GetAxisByName on page 10-584** for details of the description, scope, and motion mode. This function accesses the group function name.

```
int GetGroupAxisByName(
const char* cName
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

cName

Tag/assembly name as declared in XML configuration file.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.2.8. SetBoolParameter

Refer to the section **5.7.23 MMC_WriteBoolParameter on page 5-215** for details of the description, scope, and motion mode.

```
void SetBoolParameter(
unsigned long ulValue,
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

ulValue

Any integer value. +ve numeric value.

eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.4 Axis Status on page 5-69** for the appropriate integer parameter to be used as enumerator.

iIndex

Index array (only relevant for array situations). Any +ve integer values

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.2.9. SetParameter

Refer to the section **5.7.27 MMC_WriteParameter on page 5-227** for details of the description, scope, and motion mode.

```
void SetParameter(
double dbValue,
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

dbValue

Array parameter with double value.

eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.4 Axis Status on page 5-69** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.2.10. GetBoolParameter

Refer to the section **5.7.10 MMC_ReadBoolParameter on page 5-173** for details of the description, scope, and motion mode.

```
unsigned long GetBoolParameter(
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
)throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.4 Axis Status on page 5-69** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values

Return

lValue Boolean parameters integer value

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.2.11. GetParameter

Refer to the section **5.7.15 MMC_ReadParameter on page 5-190** for details of the description, scope, and motion mode.

```
double GetParameter(
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
)throw (CMMCException);
```

Source GMAS\includes\CPP\MMCAxis.h

.NET Definition

Function Parameters

eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3 Axis, Group, Global, Parameters** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values

Return

dbValue

Output of the specific parameter. Any Double value.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3. The CMMCSingleAxis class

The class CMMCSingleAxis wraps the single axis functions detailed in the section **5.4 Axis Status on page 5-69**. The diagram in **Figure 15-3** describes the heirarchial structure of the classes and type definitions associated with the CMMCSingleAxis.

Figure 15-3 CMMCSingleAxis class diagram

The class CMMCSingle retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

It should be noted that Private functions and their operation should be transparent to the user, and are not for general application by the user.

CMMCSingleAxis
Class
→ CMMCNode

Fields
- m_dbHomePosition
- m_eDirection
- m_eHomeBufferMode
- m_eHomeDirection
- m_eHomeSwitchMode
- m_eHomingMode
- m_fAcceleration
- m_fDeceleration
- m_fHomeAcceleration
- m_fHomeDistanceLimit
- m_fHomeTorqueLimit
- m_fHomeVelocity
- m_fJerk
- m_fVelocity
- m_ucExecute
- m_ucHomeExecute
- m_uiExecDelayMs
- m_uiHomeTimeLimit
- m_uiHomingMethod

Methods
- ~CMMCSingleAxis
- AxisLink
- AxisUnLink
- CancelPDO
- CancelVirtualEncoder
- ChangeDefaultPDOConfig
- CMMCSingleAxis (+ 2 overloads)
- ConfigPDO
- ConfigPDOEventMode
- ConfigVirtualEncoder
- ElmoCallAsync
- ElmoExecute
- ElmoGetAsyncFloatArray
- ElmoGetAsyncFloatParam
- ElmoGetAsyncIntArray
- ElmoGetAsyncIntParam
- ElmoGetReply (+ 1 overload)
- ElmoGetSyncArray (+ 1 overload)
- ElmoGetSyncParam (+ 1 overload)
- ElmoIsReplyAwaiting
- ElmoSetAsyncArray (+ 1 overload)
- ElmoSetAsyncParam (+ 1 overload)
- GetActualPosition
- GetActualTorque
- GetActualVelocity
- GetAxisError
- GetBoolParameter
- GetDigInput
- GetDigInputs
- GetDigOutputs
- GetDigOutputs32bit (+ 1 overload)
- GetOpMode
- GetParameter
- GetStatusRegister (+ 1 overload)
- Halt (+ 2 overloads)
- Home (+ 1 overload)
- HomeDS402 (+ 1 overload)
- MoveAbsolute (+ 3 overloads)
- MoveAbsoluteRepetitive (+ 4 overloads)
- MoveAdditive (+ 3 overloads)
- MoveAdditiveRepetitive (+ 4 overloads)
- MoveRelative (+ 3 overloads)
- MoveRelativeRepetitive (+ 4 overloads)
- MoveVelocity
- PositionProfile
- PowerOff
- PowerOn
- SetBoolParameter
- SetDefaultHomeDS402Params
- SetDefaultHomeParams
- SetDefaultParams
- SetDigOutputs (+ 2 overloads)
- SetDigOutputs32Bit (+ 1 overload)
- SetOpMode
- SetOverride
- SetParameter
- SetPosition
- Stop (+ 2 overloads)
- TouchProbeDisable
- TouchProbeEnable
- USleep

Figure 15-4 Fields and methods of the CMCSingleAxis class

The detailed class view shown in **Figure 15-4**, describes the fields and methods associated with the CMMCSingleAxis class. These are generally default parameters, which can be operated using their default values. However if the user wishes to change the defaults, refer to the relevant parameter section in the manual.

15.3.1. SetDefaultParams

Sets the single axis' default parameters, and overwrites the class default parameters.

```
void SetDefaultParams(
const MMC_MOTIONPARAMS_SINGLE& stSingleAxisParams
);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

stSingleAxisParams

stSingleAxisParams references the structure MMC_MOTIONPARAMS_SINGLE with default parameters, and either returns none, or throws CMMCException on failure.

MMC_MOTIONPARAMS_SINGLE Structure

```
typedef struct{
double dbPosition;
double dbDistance;
float fEndVelocity;
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_DIRECTION_ENUM eDirection ;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
unsigned int uiExecDelayMs;
}MMC_MOTIONPARAMS_SINGLE;
```

Parameters

All parameters

Refer to the **MMC_MOVEABSOLUTE_IN structure on page 5-102** and **MMC_MOVEABSOLUTEREPETITIVE_IN** structure **on page 5-126** for details of the parameters.

15.3.2. SetDefaultHomeDS402Params

Sets the default home (DS-402) parameters, and overwrites the class default parameters.

```
void SetDefaultHomeDS402Params(
const MMC_HOMEDS402_IN& stSingleParams
);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

stSingleAxisParams

stSingleAxisParams references the structure MMC_MOTIONPARAMS_SINGLE with default parameters, and either returns none, or throws CMMCException on failure.

MMC_HOMEDS402_IN Structure

```
typedef struct{
double dbPosition;
float fAcceleration;
float fVelocity;
float fDistanceLimit;
float fTorqueLimit;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned int uiHomingMethod;
unsigned int uiTimeLimit;
unsigned char ucExecute;
}MMC_HOMEDS402_IN;
```

Parameters

All parameters

Refer to the **MMC_HOMEDS402_IN Structure** for details of the parameters.

15.3.3. SetDefaultHomeParams

Sets the default home parameters, and overwrites the class default parameters.

```
void SetDefaultHomeParams(
const MMC_HOME_IN& stSingleParams
);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

MMC_HOME_IN& stSingleParams

stSingleParams references the structure MMC_HOME_IN with default parameters, and either returns none, or throws CMMCException on failure.

Refer to the section describing the function **MMC_HOME_IN Structure on page 5-82**.

15.3.4. Home

Refer to the section **5.6.2 MMC_Home on page 5-81** for details of the description, scope, and motion mode.

```
unsigned short Home(
[MMC_HOME_IN stHomeParams]
short* usErrorID,
unsigned int* uiHndl
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

MMC_HOME_IN stHomeParams

Refer to the section describing the function **MMC_HOME_IN Structure on page 5-82**.

usErrorID

Returned command error ID as -ve or +ve integers. Signals where an error has occurred within function block:

- MC_TimeLimitExceeded
- MC_DistanceLimitExceeded
- MC_TorqueLimitExceeded

Refer to the errors listed in sections **6.2 G-MAS Error IDs,** and **6.7 NC Profiler Error IDs on page 6-417- 6-453.**

uiHndl

Returned function block handle. Any +ve value.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.5. HomeDS402

Refer to the section **5.6.3 MMC_HomeDS402 on page 5-96** for details of the description, scope, and motion mode.

```
void HomeDS402(
[MMC_HOMEDS402_IN stHomeDS402Params]
) throw (CMMCException)
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

MMC_HOMEDS402_IN stHomeDS402Params

Refer to function parameter MMC_HOMEDS402_IN in section **MMC_HOMEDS402_IN Structure on page 5-96.**

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.6. MoveAbsolute

Refer to the section **5.6.4 MMC_MoveAbsolute on page 5-101** for details of the description, scope, and motion mode.

```
int MoveAbsolute(
double dPos,
[float fVel],
[float fAcceleration],
[float fDeceleration],
[float fJerk],
MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

dbPos

Target position for the motion. Any -ve or +ve double values in technical unit [u]

fVel

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s.

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

eBufferMode

Refer to the section **MMC_MOVEABSOLUTE_IN Structure 5-102**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.7. MoveAdditive

Refer to the section **5.6.5 MMC_MoveAdditive on page 5-107** for details of the description, scope, and motion mode.

```
int MoveAdditive(
double dbDistance,
[float fVel],
[float fAcceleration],
[float fDeceleration],
[float fJerk],
MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

dbDistance

Target distancefor the motion. Any -ve or +ve double values in technical unit [u]

fVel

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s.

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

eBufferMode

Refer to **MMC_MOVEADDITIVE_IN Structure on page 5-108**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.8. MoveRelative

Refer to the section **5.6.6 MMC_MoveRelative on page 5-113** for details of the description, scope, and motion mode.

```
int MoveRelative(
double dbDistance,
[float fVel],
[float fAcceleration],
[float fDeceleration],
[float fJerk],
MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

dbDistance

Target distancefor the motion. Any -ve or +ve double values in technical unit [u]

fVel

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s.

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

eBufferMode

Refer to **MMC_MOVERELATIVE_IN Structure on page 5-114**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.9. MoveVelocity

Refer to the section **5.6.7 MMC_MoveVelocity on page 5-119** for details of the description, scope, and motion mode.

```
int MoveVelocity(
float fVelocity,
MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

fVelocity

Value of the maximum velocity (not necessarily reached). Any -ve or +ve double values in technical unit [u].

eBufferMode

Refer to **MMC_MOVEVELOCITY_IN Structure on page 5-120**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.10. MoveAbsoluteRepetitive

Refer to the section **5.6.8 MMC_MoveAbsoluteRepetitive on page 5-125** for details of the description, scope, and motion mode.

```
int MoveAbsoluteRepetitive(
double dPos,
[float fVel],
[float fAcceleration],
[float fDeceleration],
[float fJerk],
[unsigned int uiExecDelayMs],
MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

dbPos

Target position for the motion. Any -ve or +ve double values in technical unit [u]

fVel

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s.

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

uiExecDelayMs

The delay in execution of the next action (in msecs). Any +ve integer value.

eBufferMode

Refer to the **MMC_MOVEABSOLUTEREPETITIVE_IN Structure on page 5-126**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.3.11. MoveRelativeRepetitive

Refer to the section **5.6.9 MMC_MoveRelativeRepetitive on page 5-130** for details of the description, scope, and motion mode.

```
int MoveRelativeRepetitive(
double dPos,
[float fVel],
[float fAcceleration],
[float fDeceleration],
[float fJerk],
[unsigned int uiExecDelayMs]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

dbPos

Target position for the motion. Any -ve or +ve double values in technical unit [u]

fVel

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s.

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

uiExecDelayMs

The delay in execution of the next action (in msecs). Any +ve integer value.

eBufferMode

Refer to the **MMC_MOVERELATIVEREPETITIVE_IN Structure on page 5-131**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.3.12. MoveAdditiveRepetitive

Refer to the section **5.6.10 above MMC_MoveAdditiveRepetitive on page 5-135** for details of the description, scope, and motion mode.

```
int MoveAdditiveRepetitive(
double dPos,
[float fVel],
[float fAcceleration],
[float fDeceleration],
[float fJerk],
[unsigned int uiExecDelayMs]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

dbPos

Target position for the motion. Any -ve or +ve double values in technical unit [u]

fVel

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s.

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in $u/s^{2.}$

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Maximum float value of the Jerk. Any positive value in u/s^3

uiExecDelayMs

The delay in execution of the next action (in msecs). Any +ve integer value.

eBufferMode

Refer to the **MMC_MOVEADDITIVEREPETITIVE_IN Structure on page 5-136**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.3.13. PositionProfile

Refer to the section **5.7.5 MMC_PositionProfile on page 5-158** for details of the description, scope, and motion mode.

```
int PositionProfile(
MC_PATH_REF hMemHandle,
MC_BUFFERED_MODE_ENUM eBufferMode = MC_BUFFERED_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

hMemHandle

MC_PATH_REF enumerator handle to a journal entry where the pointer to the shared memory is located. MC_PATH_REF is the journal entry path reference.

hMemHandle can have integer values.

fVel

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s.

eBufferMode

Refer to the **MMC_POSITIONPROFILE_IN Structure on page 5-159.**

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.14. TouchProbeDisable

Refer to the section **5.7.22 MMC_TouchProbeDisable on page 5-212** for details of the description, scope, and motion mode.

```
int TouchProbeDisable(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.15. TouchProbeEnable

Refer to the section **5.7.21 MMC_TouchProbeEnable on page 5-209** for details of the description, scope, and motion mode.

```
int TouchProbeEnable(
unsigned char ucTriggerType
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

ucTriggerType

Trigger enables the touch proble. Reference to the trigger signal source, where the trigger input may be specified by the AXIS_REF. Has the following enumerator values:

eMMC_TOUCHPROBE_POS_EDGE = 0,

eMMC_TOUCHPROBE_NEG_EDGE

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.16. SetOpMode

Changes the motion mode between NC and Distributed. This is previous determined in the DS-402 mode. Refer to the similar function block described in section **11.5.9 MMC_ChngOpMode on page 11-772** for details of the description, scope, and motion mode.

```
int SetOpMode(
OPM402 eMode
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

OPM402 eMode

Refer to the section **11.4.2 CANopen DS-402 Modes of Operation on page 11-737** for further details and similar useage of the parameter *ucMotionMode*.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.17. GetOpMode

Changes the motion mode between NC and Distributed. This is previous determined in the DS-402 mode. Refer to the similar function block described in section **11.5.9 MMC_ChngOpMode on page 11-772** for details of the description, scope, and motion mode.

```
OPM402 GetOpMode(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.18. PowerOn

Sets the drive to Power On mode. Refer to the similar function block described in section **5.7.3 MMC_GetTotalFbDepth** for details of the description, scope, and motion mode.

```
void PowerOn(
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

eBufferMode

Refer to the **MMC_POWER_IN Structure on page 5-155**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.19. PowerOff

Sets the drive to Power Off mode. Refer to the similar function block described in section **5.7.3 MMC_GetTotalFbDepth** for details of the description, scope, and motion mode.

```
void PowerOff(
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

eBufferMode

Refer to the **MMC_POWER_IN Structure on page 5-155**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including as for PowerOn:
Function Name
Structure name
Axis reference
Error ID
Status of the axis

15.3.20. GetActualPosition

Refer to the similar function block described in section **5.7.6 MMC_ReadActualPosition on page 5-161** for details of the description, scope, and motion mode.

```
double GetActualPosition(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.21. GetActualVelocity

Refer to the similar function block described in section **5.7.8 MMC_ReadActualVelocity on page 5-167** for details of the description, scope, and motion mode.

```
double GetActualVelocity(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.22. GetActualTorque

Refer to the similar function block described in section **5.7.7 MMC_ReadActualTorque on page 5-164** for details of the description, scope, and motion mode.

```
double GetActualTorque(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.3.23. Halt

Refer to the section **5.6.1 MMC_Halt on page 5-77** for details of the description, scope, and motion mode.

```
void Halt(
[float fDeceleration],
[float fJerk],
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

Refer to the **MMC_HALT_IN Structure on page 5-78**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.3.24. Stop

Refer to the section **5.6.11 MMC_Stop on page 5-140** for details of the description, scope, and motion mode.

```
void Stop(
[float fDeceleration],
[float fJerk],
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

Refer to the **MMC_STOP_IN Structure on page 5-141**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.25. GetAxisError

Refer to the section **5.7.9 MMC_ReadAxisError on page 5-170** for details of the description, scope, and motion mode.

```
unsigned short GetAxisError(
unsigned short* usLastEmergencyErrCode
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

usLastEmergencyErrCode

Last emergency DS-402 error code that occurred in the system. Refer to the DS-402 Elmo emergency codes in the following:

- CANopen DS-301 Implementation Guide Chapter 6
- CANopen DSP-402 Implementation Guide

SimplIQ Command Reference Guide or Gold Command Reference Guide

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name
Axis reference Error ID
Status of the axis.

15.3.26. GetDigInput[s]

Refer to the section **5.7.12 MMC_ReadDigitalInput(s) on page 5-179** for details of the description, scope, and motion mode.

```
unsigned char GetDigInput[s](
[int iInputNumber]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

iInputNumber

Selects the input depending on the drive. Can be part of MMC_Axis_Ref, if only one single input is referenced. +ve integer value.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name
Axis reference Error ID
Status of the axis.

15.3.27. GetDigOutputs32Bit

Refer to the section **5.7.14 MMC_ReadDigitalOutputs32Bit on page 5-187** for details of the description, scope, and motion mode.

```
unsigned long GetDigOutputs32bit(
int iOutputNumber
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

iOutputNumber

Selects the output. Can be part of MC_OutputRef, if only one single output is referenced. +ve integer value

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.3.28. GetDigOutputs

Refer to the section **5.7.13 MMC_ReadDigitalOutputs on page 5-184** for details of the description, scope, and motion mode.

```
unsigned char GetDigOutputs(
int iOutputNumber
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

iOutputNumber

Selects the output. Can be part of MC_OutputRef, if only one single output is referenced. +ve integer value

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.3.29. SetDigOutputs32Bit

Refer to the section **5.7.26 MMC_WriteDigitalOutputs32Bit on page 5-224** for details of the description, scope, and motion mode.

```
void SetDigOutputs32Bit(
[const int iOutputNumber,]
const unsigned long ulValue
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

iOutputNumber

Selects the output. Can be part of MC_OutputRef, if only one single output is referenced. +ve integer value

ulValue

Total value of all the inputs together. +ve 32 bit bitwise numeric value.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name	
Axis reference	Error ID	Status of the axis.

15.3.30. SetDigOutputs

Refer to the section **5.7.25 MMC_WriteDigitalOutputs on page 5-221** for details of the description, scope, and motion mode.

```
void SetDigOutputs(
[const int iOutputNumber,]
const unsigned char ucValue
[unsigned char ucEnable]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

iOutputNumber

Selects the output. Can be part of MC_OutputRef, if only one single output is referenced. +ve integer value

ucValue

Selected value of the input signal. +ve integer value

ucEnable

Get the value of the parameter continuously while enabled. As long as Enable is true, power is enabled. Character values of 0, 1 integers.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name Axis reference
Error ID Status of the axis.

15.3.31. SetOverride

Refer to the section **5.7.19 MMC_SetOverride on page 5-202** for details of the description, scope, and motion mode.

```
void SetOverride(
float fAccFactor,
float fJerkFactor,
float fVelFactor
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

fAccFactor

New override factor for the acceleration/deceleration. Any +ve float value between [0 – 1].

fJerkFactor

New override factor for the jerk. Any +ve float value between [0 – 1].

fVelFactor

New override factor for the velocity. Any +ve float value between [0 – 1].

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**.

15.3.32. ConfigPDO

Refer to the sections **11.5.4 - 11.5.7 on page 11-752 - 11-764** for details of the description, scope, and motion mode.

```
void ConfigPDO(
PDO_NUMBER_ENUM ePDONum,
PDO_PARAM_TYPE_ENUM eParamType,
unsigned int uiPDOCommParamEvent,
unsigned short usEventTimer,
unsigned char ucEventGroup,
unsigned char ucPDOCommParam,
unsigned char ucSubIndex,
unsigned char ucPDOType
)throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

ePDONum

Defines an enumerator whose structure PDO_NUMBER_ENUM is the PDO number with values:

PDO_NUM_3 = 3

PDO_NUM_4 = 4

eParamType

Defines an enumerator whose structure PDO_PARAM_TYPE_ENUM has the following values:

PDO_PARAM_REG = 1

PDO_PARAM_USE = 2

uiPDOCommParamEvent

This parameter inserts a PDO Events mechanism in the GMAS for servo drive operations like emit, motion complete, motion started etc. When the operation is performed, an event is sent back to the G-MAS, and thereon to any connected host.

Any +ve integer in bitwise form is accepted.

usEventTimer

The timer for the PDO event. Has the following conditions depending on the drive specifications. Acceptable values are any integer in millisecs:

0 for Synchronous data

>0 for Asynchronous data

For the following PDO event times:

MC_PDO_TIMER_NON = 0

MC_PDO_TIMER_1_MILISEC = 1

MC_PDO_TIMER_2_MILISEC = 2
MC_PDO_TIMER_3_MILISEC = 3
MC_PDO_TIMER_4_MILISEC = 4
MC_PDO_TIMER_5_MILISEC = 5
MC_PDO_TIMER_10_MILISEC = 10
MC_PDO_TIMER_20_MILISEC = 20
MC_PDO_TIMER_25_MILISEC = 25
MC_PDO_TIMER_50_MILISEC = 50
MC_PDO_TIMER_100_MILISEC = 100
MC_PDO_TIMER_150_MILISEC = 150
MC_PDO_TIMER_200_MILISEC = 200
MC_PDO_TIMER_250_MILISEC = 250
MC_PDO_TIMER_255_MILISEC = 255

ucEventGroup

Defines which group of events are to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738** the correct definition to be used. Any +ve character values are acceptable.

ucPDOCommParam

PDO communications parameter. Has the following +ve character values:

PDO_COM_PARAM_SYNC 0x01
PDO_COM_PARAM_ASYNC 0xFF
PDO_COM_PARAM_EVENT 0xFE

PDO events are only possible when the input argument *ucPDOCommParam*, is PDO_COM_PARAM_EVENT.

ucSubIndex

Defines which index value signifies User Integer and User Float values of the servo drive. Refer to the section **11.4.4 Using Event Groups** 16 and 17

This section explains how to use the Event Groups 16 and 17 with the functions MMC_CfgUserParamEvPDO4Cmd or MMC_CfgUserParamEvPDO3Cmd. It is important to note that MMC_CfgUserParamEvPDO4Cmd or MMC_CfgUserParamEvPDO3Cmd do not support DS-406 or DS-401 nodes. Mapping a general PDO is not possible for DS-406 devices, and with a DS-401 node, it is necessary to use the same method as in DS-402, but only with the functions MMC_ConfigGeneralTPDO3 and MMC_ConfigGeneralTPDO3. The ***ucEventType*** parameter field may hold the value of event group 16 or 17.

1. Map the desired TPDO objects by manually sending SDOs to the node. This procedure should be performed in compliance with the device user manual and according to the CANopen standards.

When using event groups 16 and 17, the GMAS will not map anything by itself, but will only store the data arriving after the user has manually mapped the PDO, in conditions that allow extraction by the user.

2. Before mapping the final TPDO objects by sending the last SDO (that triggers the node to send the TPDOs), use the functions MMC_CfgUserParamEvPDO4Cmd or MMC_CfgUserParamEvPDO3Cmd with event groups 16 or 17 and the proper communication parameter defined in the mapping process for SYNC, ASYNC or "on event". All other inputs have no importance.
3. Send the last SDO to Trigger the node to send the TPDOs
4. From now on, the GMAS saves the data which the node sends.
5. The data can be extracted using the function MMC_PDOGeneralReadCmd with the following ucPDONum parameter values:

ucPDONum Value	Action
0	Extracts the data of event group 16,
1	Extracts the data of event group 17.

6. In addition, when the TPDO arrives it is possible to receive an event with the data, refer to the use of the functions MMC_CfgEventModePDO4Cmd and MMC_CfgEventModePDO3Cmd.

Servo Drive Sub-Index **on page 11-740**. Any +ve character integers are accepted as values

ucPDOType

The direction of the PDO, according to the MC_PDO_TYPE_ENUM enumerator:

MC_PDO_TYPE_RPDO

MC_PDO_TYPE_TXPDO

This will be dependent on the value of the parameters ucEventGroup, and ucSubIndex.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name

Axis reference Error ID

Status of the axis.

15.3.33. CancelPDO

Refer to the sections **11.12.1 - 11.12.4 on page 11-957 - 11-966** for details of the description, scope, and motion mode.

```
void CancelPDO(
PDO_NUMBER_ENUM ePDONum
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

ePDONum

Defines an enumerator whose structure PDO_NUMBER_ENUM is the PDO number with values:

PDO_NUM_3 = 3

PDO_NUM_4 = 4

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**.

15.3.34. ConfigPDOEventMode

Refer to the sections **11.5.10 - 11.5.11 on page 11-775 - 11-778** for details of the description, scope, and motion mode.

```
void ConfigPDOEventMode(
unsigned char ucPDOEventMode
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

ucPDOEventMode

PDO event mode. This enumerator has the following values:

MC_PDO_EVENT_NO_NOTIF = 0,

MC_PDO_EVENT_CYCLIC_NOTIF,

MC_PDO_EVENT_IMMIDIATE_NOTIF

of which the default event type is MC_PDO_EVENT_NO_NOTIF. When using MC_PDO_EVENT_IMMIDIATE_NOTIF mode, no endian swap is created on the data.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**.

15.3.35. ChangeDefaultPDOConfig

Refer to the section **11.5.8 MMC_ChangeDefaultPDOConfiguration on page 11-769** for details of the description, scope, and motion mode.

```
void ChangeDefaultPDOConfig(
unsigned char ucPDONum,
unsigned char ucPDODir,
unsigned char ucPDOCommParam
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

ucPDONum

Changes a specific PDO's communication from sync to async and visa versa. Allowed values are 1 to 4, representing the PDO1, PDO2, PDO3, and PDO4.

ucPDODir

Changes the RX or TX specific PDO communication. Allowed values are:

RXPDO

TXPDO

ucPDOCommParam

PDO communications parameter. Has the following +ve character values:

PDO_COM_PARAM_SYNC	0x01
PDO_COM_PARAM_ASYNC	0xFF
PDO_COM_PARAM_EVENT	0xFE

PDO events are only possible when the input argument *ucPDOCommParam*, is PDO_COM_PARAM_EVENT.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.3.36. ElmoSetAsyncParam

Refer to the section **11.8.3 MMC_ElmoSetParameter on page 11-889** for details of the description, and motion mode.

```
void ElmoSetAsyncParam(
char cCmd[3],
int& iVal
[float& fVal]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iVal or fVal

Integer value of the data that is to be set.

Optionally, float value of the data.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.37. ElmoGetAsyncIntParam

Refer to the section **11.8.4 MMC_ElmoGetParameter on page 11-891** for details of the description, and motion mode.

```
void ElmoGetAsyncIntParam(
char cCmd[3]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.38. ElmoGetAsyncFloatParam

Refer to the section **11.8.4 MMC_ElmoGetParameter on page 11-891** for details of the description, and motion mode.

```
void ElmoGetAsyncFloatParam(
char cCmd[3]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.39. ElmoGetAsyncIntArray

Refer to the section **11.8.5 MMC_ElmoGetArray on page 11-893** for details of the description, and motion mode.

```
void ElmoGetAsyncIntArray(
char cCmd[3],
short iArrayIdx
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iArrayIdx

Index of array element. Any +ve or –ve short integer value with a maximum of 2 bytes.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.40. ElmoGetAsyncFloatArray

Refer to the section **11.8.5 MMC_ElmoGetArray on page 11-893** for details of the description, and motion mode.

```
void ElmoGetAsyncFloatArray(
char cCmd[3],
short iArrayIdx
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iArrayIdx

Index of array element. Any +ve or –ve short integer value with a maximum of 2 bytes.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.41. ElmoSetAsyncArray

Refer to the section **11.8.8 MMC_ElmoSetArray on page 11-899** for details of the description, and motion mode.

```
void ElmoSetAsyncArray(
char cCmd[3],
short iArrayIdx,
float& fVal
[int& iVal]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iArrayIdx

Index of array element. Any +ve or –ve short integer value with a maximum of 2 bytes.

iVal or fVal

Integer value of the data that is to be set.

Optionally, float value of the data.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.42. ElmoGetSyncParam

Refer to the section **11.8.4 MMC_ElmoGetParameter on page 11-891** for details of the description, and motion mode.

```
void ElmoGetSyncParam(
char cCmd[3],
float& fVal
[int& iVal]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iVal or fVal

Integer value of the data that is to be set.

Optionally, float value of the data.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.43. ElmoGetSyncArray

Refer to the section **11.8.5 MMC_ElmoGetArray on page 11-893** for details of the description, and motion mode.

```
void ElmoGetSyncArray(
char cCmd[3],
short iArrayIdx,
float& fVal
[int& iVal]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iArrayIdx

Index of array element. Any +ve or –ve short integer value with a maximum of 2 bytes.

iVal or fVal

Integer value of the data that is to be set.

Optionally, float value of the data.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.44. ElmoCallAsync

Refer to the section **11.8.12 MMC_ElmoCall on page 11-905** for details of the description, and motion mode.

```
void ElmoCallAsync(
char cCmd[3]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.45. ElmoExecute

Refer to the section **11.8.2 MMC_ElmoExecuteLabel on page 11-886** for details of the description, and motion mode.

```
void ElmoExecute(
unsigned char* pData,
unsigned char ucLength
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

pData

String Data with the precursor format of [Metronome command]##[label]. Any +ve character values with a maximum length of 80 bytes.

ucLength

Length of the label string. Length with the precursor format of [Metronome command]##[label]. Any +ve character values.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.46. ElmoIsReplyAwaiting

Refer to the section **11.8.9 MMC_ElmoQueryOperationFIFOIndex on page 11-901** for details of the description, and motion mode.

```
int ElmoIsReplyAwaiting(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**.

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.47. ElmoGetReply

Refer to the section **11.8.10 MMC_ElmoQueryOperationFIFORetrieveData on page 11-902** for details of the description, and motion mode.

```
void ElmoGetReply(
float& fVal
[int& iVal]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

iVal or fVal

Integer value of the data that is to be set.

Optionally, float value of the data.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

Scope

Note: **When using the binary interpreter, first open a UDP channel. Otherwise, the binary interpreter functions will return error -10(Lib error).**

15.3.48. ConfigVirtualEncoder

Refer to the section **11.5.12 MMC_ConfigVirtualEncoder on page 11-781** for details of the description, scope, and motion mode.

```
void ConfigVirtualEncoder(
double dbLowPos,
double dbHighPos,
float fFactor,
unsigned char ucMode,
unsigned char ucGroupID
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

dbLowPos

Low range of the virtual encoder. Any -ve or +ve double values in technical unit [u]

dbHighPos

High range of the virtual encoder. Any -ve or +ve double values in technical unit [u]

fFactor

Encoder factor. Any +ve integer value accepted.

ucMode

Defines the virtual encoder mode of the encoder with the following options:

NC_NODE_VIRTUAL_ENC_MODE_DISABLED = 0

NC_NODE_VIRTUAL_ENC_MODE_TARGET_POS

NC_NODE_VIRTUAL_ENC_MODE_ACTUAL_POS

ucGroupID

Group CAN ID. +ve integer accepted.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.3.49. CancelVirtualEncoder

Refer to the section **11.5.1 MMC_CancelVirtualEncoder on page 11-744** for details of the description, scope, and motion mode.

```
void CancelVirtualEncoder(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.3.50. SetPosition

Refer to the section **5.7.20 MMC_SetPosition on page 5-206** for details of the description, scope, and motion mode.

```
void SetPosition(
double dbPosition,
double dbModulus,
unsigned char ucPosMode
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

dbPosition

Target position for the motion when conditions are met. Any -ve or +ve double values in technical unit [u].

dbModulus

The relative modulus of the axis. Any -ve or +ve double values in technical unit [u].

ucPosMode

Boolean values for the position mode can be absolute mode = 0, or relative mode = 1

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.3.51. SetParameter

Refer to the section **5.7.27 MMC_WriteParameter** for details of the description, scope, and motion mode.

```
void SetParameter(
double dbValue,
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

dbValue

Array parameter with double value.

eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.4 Axis Status on page 5-69** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.52. SetBoolParameter

Refer to the section **5.7.23 MMC_WriteBoolParameter** for details of the description, scope, and motion mode.

```
void SetBoolParameter(
long lValue,
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

lValue

Input value. Any +ve integer.

eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.4 Axis Status on page 5-69** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name
Structure name
Axis reference
Error ID
Status of the axis.

15.3.53. AxisLink

Refer to the section **5.7.29 MMC_AxisLink** for details of the description, scope, and motion mode.

```
void AxisLink(
unsigned short usAxisRef,
unsigned char ucMode = 0
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

usAxisRef

Axis reference. Any +ve bitwise integer.

ucMode = 0

Defines the virtual encoder mode of the encoder with value 0. This parameter has the following options:

NC_NODE_VIRTUAL_ENC_MODE_DISABLED = 0

NC_NODE_VIRTUAL_ENC_MODE_TARGET_POS

NC_NODE_VIRTUAL_ENC_MODE_ACTUAL_POS

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.3.54. AxisUnLink

Refer to the section **5.7.30 MMC_AxisUnLink** for details of the description, scope, and motion mode.

```
void AxisUnLink(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.3.55. GetBoolParameter

Refer to the section **5.7.23 MMC_WriteBoolParameter** for details of the description, scope, and motion mode.

```
long GetBoolParameter(
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3 Axis, Group, Global, Parameters** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.3.56. GetParameter

Refer to the section **5.7.27 MMC_WriteParameter** for details of the description, scope, and motion mode.

```
double GetParameter(
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCSingleAxis.h

.NET Definition

Function Parameters

eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.4 Axis Status on page 5-69** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4. The CMMCGroupAxis class

The class CMMCGroupAxis wraps the multiple axes functions detailed in the section **5.8 Multiple Axes Motion Control**. The diagram in **Figure 15-5** describes the heirarchial structure of the classes and type definitions associated with the CMMCGroupAxis.

Figure 15-5 CMMCGroupAxis class diagram

The class CMMCGroupAxis retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

It should be noted that Private functions and their operation should be transparent to the user, and are not for general application by the user.

Figure 15-6 Fields and methods of the CMCGroupAxis class

The detailed class view shown in **Figure 15-6**, describes the fields and methods associated with the CMMCGroupAxis class. These are generally default parameters, which can be operated using their default values. However if the user wishes to change the defaults, refer to the relevant parameter section in the manual.

15.4.1. SetDefaultParams

Sets the multiple axes' default parameters, and overwrites the class default parameters.

```
void SetDefaultParams(
const MMC_MOTIONPARAMS_GROUP& stGroupAxisParams
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

stGroupAxisParams

stGroupAxisParams references the structure MMC_MOTIONPARAMS_GROUP with default parameters, and either returns none, or throws CMMCException on failure.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

MMC_MOTIONPARAMS_GROUP Structure

```
typedef struct{
double dAuxPoint[NC_MAX_NUM_AXES_IN_NODE];
double dEndPoint[NC_MAX_NUM_AXES_IN_NODE];
float fVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE];
MC_COORD_SYSTEM_ENUM eCoordSystem;
NC_TRANSITION_MODE_ENUM eTransitionMode;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucSuperimposed;
unsigned int m_uiExecDelayMs;
unsigned char ucExecute;
}MMC_MOTIONPARAMS_GROUP;
```

Parameters

All parameters

Refer to the **MMC_MOVECIRCULARABSOLUTE_IN** structure **on page 5-284** and **MMC_MOVELINEARABSOLUTEREPETITIVE_IN** structure **on page 5-340** for details of the parameters.

15.4.2. SetKinTransform

Sets the multiple axes' default parameters, and overwrites the class default parameters.

```
void SetKinTransform(
MMC_SETKINTRANSFORM_IN& stInParam
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

stInParam

stInParam references the structure MMC_SETKINTRANSFORM_IN with default parameters, and either returns none, or throws CMMCException on failure.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

MMC_SETKINTRANSFORM_IN Structure

Parameters

All parameters

Refer to the section **MMC_SETKINTRANSFORM_IN on page 5-399** for details of the parameters.

15.4.3. RemoveAxisFromGroup

Refer to the section **5.9.11 MMC_RemoveAxisFromGroup on page 5-395** for details of the description, scope, and motion mode.

```
void RemoveAxisFromGroup(
NC_IDENT_IN_GROUP_ENUM eIdentInGroup
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

eIdentInGroup

The NC_IDENT_IN_GROUP_ENUM enumerator identifies the order and Nodes in the group of the added axis. Performed via an enumerator to give the different axes a name in the order, which can be coupled to the names in the kinematic model. The options are:

NC_NODE_1_ID = 0

……

NC_NODE_16_ID = 15

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.4. MoveCircularAbsolute

Refer to the section **5.8.8 MMC_MoveCircularAbsolute on page 5-283** for details of the description, scope, and motion mode.

```
int MoveCircularAbsolute(
NC_ARC_SHORT_LONG_ENUM eArcShortLong,
NC_PATH_CHOICE_ENUM ePathChoice,
NC_CIRC_MODE_ENUM eCircleMode,
[double dAuxPoint[]]
[double dEndPoint[]]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

eArcShortLong

Defines the types of supported arc length. The NC_ARC_SHORT_LONG_ENUM enumerator options are:

MC_NONE_ARC_CHOICE = 0

MC_SHORT = 1

MC_LONG = 2

ePathChoice

Defines the NC_PATH_CHOICE_ENUM enumerator types of supported path choice. The option are:

MC_NONE_PATH_CHOICE = 0

MC_CLOCKWISE = 1

MC_COUNTERCLOCKWISE = 2

eCircleMode

Defines the types of supported circular modes in 2D. Refer to the section **5.8.2 Transition and Circular Modes**, and the definitions below.

The NC_CIRC_MODE_ENUM enumerator options are:

MC_NONE_CIRC_MODE = 0

MC_BORDER_CIRC_MODE= 1

MC_CENTER_CIRC_MODE = 2

MC_RADIUS_CIRC_MODE = 3

MC_ANGLE_CIRC_MODE = 4

dAuxPoint[]

Absolute position for a dimension in the coordinate system specified by the input signal CoordSystem.

dAuxPoint can have double values in a technical unit [u].

dEndPoint[]

Absolute end point position for a dimension in the coordinate system specified by the input signal CoordSystem. *dEndPoint* is a 2D or 3D double in technical unit [u].

eBufferMode

Refer to the structure **MMC_MOVECIRCULARABSOLUTE_IN on page 5-284** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.4.5. MoveCircularAbsoluteCenter

Refer to the section **5.8.9 MMC_MoveCircularAbsoluteCenter on page 5-291** for details of the description, scope, and motion mode.

```
int MoveCircularAbsoluteCenter(
NC_ARC_SHORT_LONG_ENUM eArcShortLong,
double dBorderPoint[]
[double dCenterPoint[]]
[double dEndPoint[]]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

eArcShortLong

Defines the types of supported arc length. The NC_ARC_SHORT_LONG_ENUM enumerator options are:

MC_NONE_ARC_CHOICE = 0

MC_SHORT = 1

MC_LONG = 2

dBorderPoint

Absolute border position for a dimension in the coordinate system specified by the input signal CoordSystem. *dBorderPoint* can have double values in a technical unit [u].

dCenterPoint

Absolute position for a dimension in the coordinate system specified by the input signal CoordSystem. *dCenterPoint* can have double values in a technical unit [u].

dEndPoint

Absolute end point position for a dimension in the coordinate system specified by the input signal CoordSystem.*dEndPoint* is a 2D or 3D double vector in technical unit [u].

eBufferMode

Refer to the structure **MMC_MOVECIRCULARABSOLUTECENTER_IN Structure on page 5-292** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name

Axis reference Error ID

Status of the axis.

15.4.6. MoveCircularAbsoluteBorder

Refer to the section **5.8.10 MMC_MoveCircularAbsoluteBorder on page 5-297** for details of the description, scope, and motion mode.

```
int MoveCircularAbsoluteBorder(
double dBorderPoint[],
[double dEndPoint[],]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

dBorderPoint

Absolute border position for a dimension in the coordinate system specified by the input signal CoordSystem. *dBorderPoint* can have double values in a technical unit [u].

dEndPoint

Absolute end point position for a dimension in the coordinate system specified by the input signal CoordSystem.*dEndPoint* is a 2D or 3D double vector in technical unit [u].

eBufferMode

Refer to the structure **MMC_MOVECIRCULARABSOLUTEBORDER_IN Structure on page 5-298** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.7. MoveCircularAbsoluteRadius

Refer to the section **5.8.11 MMC_MoveCircularAbsoluteRadius on page 5-303** for details of the description, scope, and motion mode.

```
int MoveCircularAbsoluteRadius(
NC_ARC_SHORT_LONG_ENUM eArcShortLong,
NC_PATH_CHOICE_ENUM ePathChoice,
[double dSpearHeadPoint[],]
[double dEndPoint[],]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

eArcShortLong

Defines the types of supported arc length. The NC_ARC_SHORT_LONG_ENUM enumerator options are:

MC_NONE_ARC_CHOICE = 0

MC_SHORT = 1

MC_LONG = 2

ePathChoice

Defines the NC_PATH_CHOICE_ENUM enumerator types of supported path choice. The option are:

MC_NONE_PATH_CHOICE = 0

MC_CLOCKWISE = 1

MC_COUNTERCLOCKWISE = 2

dSpearHeadPoint

Absolute radius position for a dimension in the coordinate system specified by the input signal CoordSystem. *dSpearHeadPoint* can have double values in a technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

dEndPoint[]

Absolute end point position for a dimension in the coordinate system specified by the input signal CoordSystem. *dEndPoint* is a 2D or 3D double in technical unit [u].

eBufferMode

Refer to the structure **MOVECIRCULARABSOLUTERADIUS_IN Structure on page 5-304** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.4.8. MoveCircularAbsoluteAngle

Refer to the section **5.8.11 MMC_MoveCircularAbsoluteRadius on page 5-303** for details of the description, scope, and motion mode.

```
int MoveCircularAbsoluteAngle(
[double dAngle,]
[double dCenterPoint[],]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

dAngle

Absolute angular position for the coordinate system specified by the input signal CoordSystem. Angular double value in degrees [u]

dCenterPoint

Absolute position for a dimension in the coordinate system specified by the input signal CoordSystem. *dCenterPoint* can have double values in a technical unit [u].

eBufferMode

Refer to the structure **MOVECIRCULARABSOLUTERADIUS_IN Structure on page 5-304** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.4.9. MoveLinearAbsolute

Refer to the section **5.8.13 MMC_MoveLinearAbsolute on page 5-315** for details of the description, scope, and motion mode.

```
int MoveLinearAbsolute(
[float fVelocity,]
[double dbPosition[NC_MAX_NUM_AXES_IN_NODE],]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

fVelocity

Value of the maximum velocity (not necessarily reached). Any -ve or +ve double values in technical unit [u].

dbPosition

Target position for the motion of the axis when conditions are met. Any -ve or +ve double values in technical unit [u].

Array of coordinates, incl. positions and orientations (Distance if Mode = RELATIVE). The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eBufferMode

Refer to the structure **MMC_MOVELINEARABSOLUTE_IN Structure on page 5-316** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.4.10. MoveLinearRelative

Refer to the section **5.8.14 MMC_MoveLinearRelative on page 5-322** for details of the description, scope, and motion mode.

```
int MoveLinearRelative(
[float fVelocity,]
[double dbDistance[NC_MAX_NUM_AXES_IN_NODE],]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

fVelocity

Value of the maximum velocity (not necessarily reached). Any -ve or +ve double values in technical unit [u].

dbDistance

Array [1..N] of relative distances for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbDistance is a double vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eBufferMode

Refer to the structure **MMC_MOVELINEARRELATIVE_IN Structure on page 5-323** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.4.11. GroupSetOverride

Refer to the section **5.9.9 MMC_GroupSetOverride on page 5-386** for details of the description, scope, and motion mode.

```
int GroupSetOverride(
float fVelFactor,
float fAccFactor,
float fJerkFactor,
unsigned short usUpdateVelFactorIdx
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

fVelFactor

New override factor for the velocity. Any +ve float value between [0 – 1].

fAccFactor

New override factor for the acceleration/deceleration. Any +ve float value between [0 – 1].

fJerkFactor

New override factor for the jerk. Any +ve float value between [0 – 1].

usUpdateVelFactorIdx

Index of changed velocity factor. Vendor defined. The default is 0. Has integer values of 0 - 2

This variable is not in use at this moment.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.12. GroupSetPosition

Refer to the section **5.9.10 MMC_GroupSetPosition on page 5-391** for details of the description, scope, and motion mode.

```
int GroupSetPosition(
double dbPosition[],
MC_COORD_SYSTEM_ENUM eCoordSystem,
unsigned char ucMode,
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

dbPosition

Target position for the motion of the axis when conditions are met. Any -ve or +ve double values in technical unit [u].

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

ucMode

RELATIVE =True, ABSOLUTE = False (Default)

RELATIVE means that Position is added to the actual position value of the axis at the time of execution. This results in a recalibration by a specified distance. ABSOLUTE means that the actual position value of the axis is set to the value specified in the Position parameter.

Values accepted are Boolean, TRUE/FALSE.

eBufferMode

Refer to the structure **MMC_GROUPSETPOSITION_IN Structure on page 5-392** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name

Axis reference Error ID

Status of the axis.

15.4.13. GroupReadStatus

Refer to the section **5.9.7 MMC_GroupReadStatus on page 5-380** for details of the description, scope, and motion mode.

```
unsigned long GroupReadStatus(
unsigned short& usGroupErrorID
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

usGroupErrorID

Returned command group error ID. Signals where an group error has occurred within function block. These values are vendor specific. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**.

Displays an error code integer.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.4.14. GroupEnable

Refer to the section **5.9.3 MMC_GroupEnable on page 5-368** for details of the description, scope, and motion mode.

```
void GroupEnable(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.4.15. GroupDisable

Refer to the section **5.9.2 MMC_GroupDisable on page 5-365** for details of the description, scope, and motion mode.

```
void GroupDisable(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.16. GroupReset

Refer to the section **5.9.8 MMC_GroupReset on page 5-383** for details of the description, scope, and motion mode.

```
void GroupReset(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.17. GroupReadActualVelocity

Refer to the section **5.9.5 MMC_GroupReadActualVelocity on page 5-374** for details of the description, scope, and motion mode.

```
double GroupReadActualVelocity(
MC_COORD_SYSTEM_ENUM eCoordSystem,
[double dVelocity[]]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

ePathChoice

Defines the NC_PATH_CHOICE_ENUM enumerator types of supported path choice. The option are:

MC_NONE_PATH_CHOICE = 0

MC_CLOCKWISE = 1

MC_COUNTERCLOCKWISE = 2

dVelocity[]

Current velocity of the group:

- For ACS the velocities of the different axes
- For MCS it provides the velocity of the TCP

dVelocity any -ve or +ve array double value in the axis's unit [u/s].

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name

Axis reference Error ID

Status of the axis.

15.4.18. GroupReadError

Refer to the section **5.9.6 MMC_GroupReadError on page 5-377** for details of the description, scope, and motion mode.

```
unsigned short GroupReadError(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.19. AddAxisToGroup

Refer to the section **5.9.1 MMC_AddAxisToGroup on page 5-362** for details of the description, scope, and motion mode.

```
void AddAxisToGroup(
NC_NODE_HNDL_T hNode,
NC_IDENT_IN_GROUP_ENUM eIdentInGroup
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

hNode

The NC_NODE_HNDL_T enumerator defines the Node handle transition. The axis ref parameter.

hNode can have any positive numeric value.

eIdentInGroup

The NC_IDENT_IN_GROUP_ENUM enumerator identifies the order and Nodes in the group of the added axis. Performed via an enumerator to give the different axes a name in the order, which can be coupled to the names in the kinematic model. The options are:

NC_NODE_1_ID = 0

……

NC_NODE_16_ID = 15

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.20. GroupReadActualPosition

Refer to the section **5.9.4 MMC_GroupReadActualPosition on page 5-371** for details of the description, scope, and motion mode.

```
int GroupReadActualPosition(
MC_COORD_SYSTEM_ENUM eCoordSystem,
double dbPosition[]
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

dbPosition[]

Target position for the motion of the axis when conditions are met. Any -ve or +ve double values in technical unit [u].

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name

Axis reference Error ID

Status of the axis.

15.4.21. GroupStop

Refer to the section **5.8.6 MMC_GroupStop on page 5-275** for details of the description, scope, and motion mode.

```
void GroupStop(
float fDeceleration,
float fJerk,
MC_BUFFERED_MODE_ENUM eBufferMode
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

Refer to the structure **MMC_GROUPSTOP_IN Structure on page 5-276** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.22. GroupHalt

Refer to the section **5.8.7 MMC_GroupHalt on page 5-279** for details of the description, scope, and motion mode.

```
void GroupHalt(
float fDeceleration,
float fJerk,
MC_BUFFERED_MODE_ENUM eBufferMode
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

Refer to the structure **MMC_GROUPHALT_IN Structure on page 5-280** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.23. MoveLinearAbsoluteRepetitive

Refer to the function block in section **5.8.17 MMC_MoveLinearAbsoluteRepetitive on page 5-339** for details of the description, scope, and motion mode.

```
int MoveLinearAbsoluteRepetitive(
[float fVelocity,]
[double dbPosition[NC_MAX_NUM_AXES_IN_NODE],]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

fVelocity

Value of the maximum velocity (not necessarily reached). Any -ve or +ve double values in technical unit [u].

dbPosition

Target position for the motion of the axis when conditions are met. Any -ve or +ve double values in technical unit [u].

Array of coordinates, incl. positions and orientations (Distance if Mode = RELATIVE). The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eBufferMode

Refer to the **MMC_MOVELINEARABSOLUTEREPETITIVE_IN Structure on page 5-340**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.4.24. MoveLinearRelativeRepetitive

Refer to **5.8.18 MMC_MoveLinearRelativeRepetitive on page 5-344** for details of the description, scope, and motion mode.

```
int MoveLinearRelativeRepetitive(
[float fVelocity,]
[double dbDistance[NC_MAX_NUM_AXES_IN_NODE],]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

fVelocity

Value of the maximum velocity (not necessarily reached). Any -ve or +ve double values in technical unit [u].

dbDistance

Array [1..N] of relative distances for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbDistance is a double vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eBufferMode

Refer to the **MMC_MOVELINEARRELATIVEREPETITIVE_IN Structure on page 5-345**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.4.25. MoveLinearAdditive

Refer to the sections **5.8.15 MMC_MoveLinearAdditive** and **5.8.16 MMC_MoveLinearAdditiveEx on page 5-329 - 5-334** for details of the description, scope, and motion mode. The MMC_MoveLinearAdditiveEx parameter is for further accuracy in setting the parameters. The double parmeters allow setting of an 8 bit value.

```
int MoveLinearAdditive(
[float fVelocity,]
[double dbDistance[NC_MAX_NUM_AXES_IN_NODE],]
MC_BUFFERED_MODE_ENUM eBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

fVelocity

Value of the maximum velocity (not necessarily reached). Any -ve or +ve double values in technical unit [u].

dbDistance

Array [1..N] of relative distances for each dimension in the specified coordinate system, with N being vendor specific. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

dbDistance is a double vector array in technical unit [u].

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eBufferMode

Refer to the **MMC_MOVELINEARADDITIVE_IN Structure on page 5-330**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.26. MovePath

Refer to section **5.8.21 MMC_MovePath** for details of the description, scope, and motion mode.

```
int MovePath(
MC_PATH_REF hMemHandle,
float fTransitionParameter[NC_MAX_NUM_AXES_IN_NODE],
MC_BUFFERED_MODE_ENUM eBufferModeeBufferMode = MC_ABORTING_MODE
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

hMemHandle

MC_PATH_REF enumerator handle to a journal entry where the pointer to the shared memory is located. MC_PATH_REF is the journal entry path reference.

hMemHandle has integer values.

fTransitionParameter

Depending on the transition mode, different supplier specific transition parameters can be used which characterize the contour curve. The array parameter NC_MAX_NUM_AXES_IN_NODE is limited to 16, and defined as the maximum number of axis in a group.

fTransitionParameter can have any positive float value in appropriate units, dependant on the TransitionMode parameter. Refer to the section ***5.8.2.1 Transition Modes***.

[NC_MAX_NUM_AXES_IN_NODE] is an array of values [2….15].

eBufferMode

Refer to the **MMC_MOVEPATH_IN Structure on page 5-354** for further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.27. PathDeselect

Refer to **5.8.21 MMC_PathUnselect on page 5-358** for details of the description, scope, and motion mode.

```
int PathDeselect(
MC_PATH_REF hMemHandle
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

hMemHandle

MC_PATH_REF enumerator handle to a journal entry where the pointer to the shared memory is located. MC_PATH_REF is the journal entry path reference.

hMemHandle has integer values.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.4.28. PathSelect

Refer to **5.8.19 MMC_PathSelect on page 5-349** for details of the description, scope, and motion mode.

```
unsigned int PathSelect(
MC_PATH_DATA_REF pPathToSplineFile
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCGroupAxis.h

.NET Definition

Function Parameters

pPathToSplineFile

This string describes where the splines data file is located. Values accepted are any characters describing a file path.

MC_PATH_DATA_REF Where the enumerator MC_PATH_DATA_REF describes the I/O definition of the path data reference using the array [NC_MAX_SPLINES_FILE_PATH_LENGTH] that defines the maximum length of the splines file path data.

MC_PATH_DATA_REF can have values of any characters.

NC_MAX_SPLINES_FILE_PATH_LENGTH can have any numeric value.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.5. The CMMCPPGlobal class

The class MMCPPGlobal wraps the global functions detailed in various sections of this API document. The diagram in **Figure 15-7** describes the heirarchial structure of the classes and type definitions associated with the MMCPPGlobal.

Figure 15-7 MMCPPGlobal class diagram

The class MMCPPGlobal retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-8 Fields and methods of the MMCPPGlobal class

The detailed class view shown in **Figure 15-8** describes the fields and methods associated with the MMCPPGlobal class. These are generally default parameters, which can be operated using their default values. However if the user wishes to change the defaults, refer to the relevant parameter section in the manual.

It should be noted that Protected and Private and Protected functions together with their operations, should be transparent to the user, and are not for general application by the user.

15.5.1. RegisterRTE

Registers the Run Time Error.

```
void RegisterRTE(
RTE_CLBKP pRTEClbk,
bool bIsToCloseOnRTE = true
)
{m_pRTEClbk = pRTEClbk;
m_bIsToCloseOnRTE = bIsToCloseOnRTE;}
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

pRTEClbk

Run Time Error callback parameter

bIsToCloseOnRTE = true

Should the operation be closed on Run Time Error? 0 or 1 option, answer is True (1)

15.5.2. RegisterWarningClbk

Registers the warning of possible error as a callback.

```
void RegisterWarningClbk(
WARNING_CLBKP pWarningClbk
)
{m_pWarningClbk =  pWarningClbk;}
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

pWarningClbk

Warning callback parameter. Boolean 0 or 1 option

15.5.3. SetThrowFlag

Set whether a throw error is flagged or not.

```
void SetThrowFlag(
bool bThrow,
bool bIsToCloseOnThrow = true
)
m_bThrowException_OnError =  bThrow;
m_bIsToCloseOnThrow = bIsToCloseOnThrow;}
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

bThrow

Use option to throw error? Boolean 0 or 1 option.

bIsToCloseOnThrow = true

If error thrown and flagged then close operation. This will be True.

15.5.4. SetThrowWarningFlag

Set whether a throw warning is flagged or not.

```
void SetThrowWarningFlag(
bool bThrowWarining
)
{m_bThrowException_OnWarning =  bThrowWarining;}
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

bThrowWarining

Use option to throw warning? Boolean 0 or 1 option.

15.5.5. SetPrintErrorFlag

Set whether a print error is flagged or not.

```
Void SetPrintErrorFlag(
bool bPrintError
)
{m_bPrintError =  bPrintError;}
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

bPrintError

Use option to print error? Boolean 0 or 1 option.

15.5.6. SetPrintWarningFlag

Set whether a print warning is flagged or not.

```
void SetPrintWarningFlag(
bool bPrintWarining
)
{m_bPrintWarning =  bPrintWarining;}
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

bPrintWarining

Use option to print warning? Boolean 0 or 1 option.

15.5.7. ThrowMessage

Details of the throw message when a function produces an error

```
Void ThrowMessage(
int iRetval,
int iErrorID,
const char* cFunctionName
);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

iRetval

Returned value of throw message. Integer value.

iErrorID

Error ID. Integer values.

cFunctionName

Name of the Function causing the error. Character value.

15.5.8. SetConnectionType

Sets the connection type globally.

```
Void SetConnectionType(
MMCPP_CONN_TYPE eConnectionType_IN
){m_eConnectionType = eConnectionType_IN;}
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

eConnectionType_IN

```
MMCPP_CONN_TYPE GetConnectionType(
)
{return m_eConnectionType;}
```

GetConnectionType

[IN] Inputs the connection type details.

15.5.9. SetMessageFileName

Sets the file name message sent to the G-MAS

```
Void SetMessageFileName(
string sMessageFileName_IN
){m_cMessage.SetMessageFileName(sMessageFileName_IN);}

void SetMessageFileName(
char* cMessageFileName_IN
){m_cMessage.SetMessageFileName(cMessageFileName_IN);}

String GetMessageFileName(){return m_cMessage.GetMessageFileName();}
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

sMessageFileName_IN

[IN] File name message. Can be string or char

cMessageFileName_IN

[IN] File name message. Can be string or char

15.5.10. GetSyncTime

Refer to the section **10.3.7 MMC_CreateSYNCTimer on page 10-552** for details of the description, scope, and communication mode.

```
unsigned short GetSyncTime(
unsigned int uiConnHndl
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

uiConnHndl

[IN] Connection handle input, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.5.11. SetSyncTime

Refer to the section **10.3.7 MMC_CreateSYNCTimer on page 10-552** for details of the description, scope, and communication mode.

```
int SetSyncTime(
unsigned int uiConnHndl,
unsigned short usSync
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

uiConnHndl

[IN] Connection handle input, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

usSync

Synchronous period setting. Any +ve value in milliseconds

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.5.12. CreateSYNCTimer

Refer to the section **10.3.7 MMC_CreateSYNCTimer on page 10-552** for details of the description, scope, and communication mode.

```
void CreateSYNCTimer(
unsigned int uiConnHndl,
MMC_SYNC_TIMER_CB_FUNC pfClbk,
unsigned short usSYNCTimerTime
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

uiConnHndl

[IN] Connection handle input, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pfClbk

[IN] Points to the callback function MMC_SYNC_TIMER_CB_FUNC using MMC_CreateSYNCTimer.

usSYNCTimerTime

[IN] Defines the time between which a synchronization message is sent as an event.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.5.13. DestroySYNCTimer

Refer to the section **10.3.8 MMC_DestroySYNCTimer on page 10-554** for details of the description, scope, and communication mode.

```
void DestroySYNCTimer(
unsigned int uiConnHndl
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

uiConnHndl

[IN] Connection handle input, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.5.14. GetConnectionReg

Obtains the connection register call back value.

```
void* GetConnectionReg(
unsigned int uiConnHndl);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

uiConnHndl

[IN] Connection handle input, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

15.5.15. ConfigBulkRead

Refer to the section **8.1.1 MMC_ConfigBulkRead on page 8-489** for details of the description, scope, and communication mode.

```
void ConfigBulkRead(
MMC_CONNECT_HNDL hConnHndl,
NC_BULKREAD_CONFIG_ENUM eConfig,
[NC_BULKREAD_PRESET_ENUM ePreset,]
[unsigned long ulParameters[],]
unsigned long ulAxisRefMask,
unsigned char ucAxisRefGroup,
float* fFactorsArray
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

hConnHndl

IN] Connection handle input, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

eConfig

Defines the reading source. eBULKREAD_CONFIG_1 is reserved to the EAS application. Acceptable values are eBULKREAD_CONFIG_1 and eBULKREAD_CONFIG_2.

NC_BULKREAD_CONFIG_ENUM defines the following values:

eBULKREAD_CONFIG_NONE = -1

eBULKREAD_CONFIG_1 = 0

eBULKREAD_CONFIG_2 = 1

eBULKREAD_CONFIG_MAX,

ePreset

Whether the preset parameters is used or not. Values accepted are 0, or 1. The enumerator NC_BULKREAD_PRESET_ENUM has values defind according to the parameter**NC_BULKREAD_PRESET_ENUM on page 8-490**.

ulParameters

Describes the user defined bulk parameters to be read. Any +ve value.

ulAxisRefMask

Defines the axes, with the bitwise information read from (within a group). For example, if this value is set to 0x03 – only axes with axisref 0 and 1 are read.

ucAxisRefGroup

Defines the "group" where the axes are located. For example, if this value is set to 0, and ulAxisRefMask value is set to 0x03 – Axes 0 and 1 will be read. If the group value is set to 1 – axes 25 and 26 will be read. Acceptable values – 0 – 3.

fFactorsArray

The *fFactorsArray* returns an array of factors for each parameter, with enumerator values:

NC_BULKREAD_PRESET_1 stOutputData[]

NC_BULKREAD_PRESET_2 stOutputData[]

NC_BULKREAD_PRESET_3 stOutputData[]

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.5.16. PerformBulkRead

Refer to the section **8.1.2 MMC_PerformBulkRead on page 8-497** for details of the description, scope, and communication mode.

```
void PerformBulkRead(
MMC_CONNECT_HNDL hConnHndl,
int iNumberOfAxes,
NC_BULKREAD_CONFIG_ENUM eConfiguration,
NC_BULKREAD_PRESET_ENUM& eChosenPreset,
NC_BULKREAD_PRESET_1 stOutputData[]
[NC_BULKREAD_PRESET_2 stOutputData[]]
[NC_BULKREAD_PRESET_3 stOutputData[]]
[NC_BULKREAD_PRESET_4 stOutputData[]]
[NC_BULKREAD_PRESET_5 stOutputData[]]
[unsigned long* ulOutputData]
);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

hConnHndl

IN] Connection handle input, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

iNumberOfAxes

The number of axes whose parameters are to be read. An integer.

eConfiguration

Refer to the definition of the input parameter in the section **MMC_PERFORMBULKREAD_IN on page 8-498**.

eChosenPreset

Refer to the definition of the output parameter in the section **MMC_PERFORMBULKREAD_OUT on page 8-498**.

stOutputData[]

Output data selection for NC_BULKREAD_PRESET_1 to 5 from the **NC_BULKREAD_PRESET_ENUM**. Refer to the section **NC_BULKREAD_PRESET_ENUM on page 8-490**.

ulOutputData

User defined output data option. Entered as unlimited set of values.

15.5.17. RegisterConnection

Registers the connection with the G-MAS.

```
int RegisterConnection(
unsigned int uiConnHndl,
void * pConn
);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

uiConnHndl

[IN] Connection handle input, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

pConn

Connection check. Boolean options 0 or 1

15.5.18. GetConnectionReg ClearConnectionReg

Obtains or clears the connection registry from the G-MAS

```
void* GetConnectionReg(
unsigned int uiConnHndl
);
void ClearConnectionReg(
unsigned int uiConnHndl
);
```

Source GMAS\includes\CPP\MMCPPGlobal.h

.NET Definition

Function Parameters

uiConnHndl

[IN] Connection handle input, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

15.6. The CMMCConnection class

The class MMCConnection wraps the global functions detailed in various sections of this API document. The diagram in **Figure 15-9** describes the heirarchial structure of the classes and type definitions associated with the MMCConnection.

Figure 15-9 MMCConnection class diagram

The class MMCConnection retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-10 Fields and methods of the MMCConnection class

The detailed class view shown in **Figure 15-10** describes the fields and methods associated with the MMCPPGlobal class. These are generally default parameters, which can be operated using their default values. However if the user wishes to change the defaults, refer to the relevant parameter section in the manual.

Figure 15-11 Events enumerators

Figure 15-11 describes the events enumerator described and called by the function RegisterEventCallback used in conjunction with the relevant type definition.

It should be noted that Protected and Private and Protected functions together with their operations, should be transparent to the user, and are not for general application by the user.

15.6.1. Event Type Definitions

The following table defines the type definitions for the events special cases. Each of the special events provides a range of parameters

Event	Parameters provided
PdoRcvEventCallback	unsigned short usAxisRef, unsigned short usGrpID, UNPDODAT unDat1 - any form of value type included in the PDO data UNPDODAT unDat2 - any form of value type included in the PDO data
HBeateEventCallback	unsigned short usAxisRef
MotionEndEventCallback	unsigned short usAxisRef, bool bResult
EmergencyEventCallback	unsigned short usAxisRef, short sEmcyCode
HomeEndedEventCallback	unsigned short usAxisRef, short sErrCode,...

Event	Parameters provided
ModbusWriteEventCallback	Values only
SysErrorEventCallback	Values only
AsyncReplyEventCallback	unsigned short usAxisRef, unsigned short usStatus, unsigned short usError, unsigned short usCobID, unsigned short usLength, unsigned char ucBuffer
TouchProbeEndCallback	unsigned short usAxisRef, long lPosition
NodeErrorEventCallback	unsigned short usAxisRef, unsigned short sErrorID, unsigned short usEmergencyCode
StopOnLimitEventCallback	unsigned short usAxisRef, unsigned short usError, unsigned int uiStatusRegister, unsigned int uiMcsLimitRegister
TableUnderflowEventCallback	unsigned short usAxisRef

15.6.2. ConnectIPC

Creates an IPC connection for WIN32 only. Any +ve integer values accepted for the connection. Refer to the the functions **10.3.25 MMC_InitConnection**, and **10.3.27 MMC_IPCInitConnection**.

```
unsigned int ConnectIPC(
int iEventMask,
MMC_CB_FUNC fpClbk
)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

iEventMask

Defined according to the event IDs described in the section **12.18 Events Enumeration on page 12-1010**. Bitwise +ve integer ID.

MMC_CB_FUNC fpClbk

The type definition MMC_CB_FUNC is part of this header file and is similar to the callback prototype described in section **12.19.1**, which also contains the fpClbk callback function.

15.6.3. ConnectIPCEx

Creates an IPC connection for WIN32 only. Any +ve integer values accepted for the connection. Refer to the the functions **10.3.25 MMC_InitConnection**, and **10.3.27 MMC_IPCInitConnection**.

```
unsigned int ConnectIPCEx(
int iEventMask,
MMC_MB_CLBK fpClbk
)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

iEventMask

Defined according to the event IDs described in the section **12.18 Events Enumeration on page 12-1010**. Bitwise +ve integer ID.

MMC_MB_CLBK fpClbk

The type definition MMC_MB_CLBK is part of this header file and is similar to the callback prototype described in section **12.19.1**, which also contains the fpClbk callback function. However, the MMC_MB_CLBK is sa specific type of callback definition used to describe functions with a variable number of parameters.

15.6.4. ConnectRPC

Creates an RPC connection. Any +ve integer values accepted for the connection. Refer to the the functions **10.3.25 MMC_InitConnection**, and **10.3.27 MMC_IPCInitConnection**.

```
unsigned int ConnectRPC(
char* cHostIP,
char* cDestIP,
int iEventMask,
MMC_CB_FUNC fpClbk
)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

cHostIP

Host IP address. Character value in the format of an IP address.

cDestIP

Destination IP address. Character value in the format of an IP address.

iEventMask

Defined according to the event IDs described in the section **12.18 Events Enumeration on page 12-1010**. Bitwise +ve integer ID.

MMC_CB_FUNC fpClbk

The type definition MMC_CB_FUNC is part of this header file and is similar to the callback prototype described in section **12.19.1**, which also contains the fpClbk callback function.

15.6.5. ConnectRPCEx

Creates an RPC connection. Any +ve integer values accepted for the connection. Refer to the the functions **10.3.25 MMC_InitConnection**, and **10.3.27 MMC_IPCInitConnection**.

```
unsigned int ConnectRPCEx(
char* cHostIP,
char* cDestIP,
int iEventMask,
MMC_MB_CLBK fpClbk
)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

cHostIP

Host IP address. Character value in the format of an IP address.

cDestIP

Destination IP address. Character value in the format of an IP address.

iEventMask

Defined according to the event IDs described in the section **12.18 Events Enumeration on page 12-1010**. Bitwise +ve integer ID.

MMC_MB_CLBK fpClbk

The type definition MMC_MB_CLBK is part of this header file and is similar to the callback prototype described in section **12.19.1**, which also contains the fpClbk callback function. However, the MMC_MB_CLBK is sa specific type of callback definition used to describe functions with a variable number of parameters.

15.6.6. SetGlobalBoolParameter

Sets the global Boolean parameters. Refer to the section **5.7.24 MMC_GlobalWriteBoolParameter** for further details.

```
void SetGlobalBoolParameter(
[long lValue,]
[double dbValue,]
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
) throw (CMMCException)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

lValue

Input parameter. Any integer value.

dbValue

Array parameter with double value.

MMC_PARAMETER_LIST_ENUM eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3 Axis, Group, Global, Parameters** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values.

Return

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.6.7. GetGlobalBoolParameter

Obtain the global Boolean parameters. Refer to the section **5.7.24 MMC_GlobalWriteBoolParameter** for further details.

```
long GetGlobalBoolParameter(
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
) throw (CMMCException)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

MMC_PARAMETER_LIST_ENUM eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3 Axis, Group, Global, Parameters** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values.

Return

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.6.8. GetGlobalParameter

Obtain the global parameters. Refer to the section **5.7.28 MMC_GlobalWriteParameter** for further details.

```
double GetGlobalParameter(
MMC_PARAMETER_LIST_ENUM eNumber,
int iIndex
) throw (CMMCException)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

MMC_PARAMETER_LIST_ENUM eNumber

Number of the parameter. One can also use symbolic parameter names, which are declared as VAR CONST.

Refer to the section **5.3 Axis, Group, Global, Parameters** for the appropriate integer parameter to be used as enumerator.

iIndex

Array index parameter (only relevant for array situations). Any +ve integer values.

Return

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.6.9. SetIsToLoadGlobalParams

Defines a flag whether to load or not, the global parameters, when updating the global parameters from a file to the G-MAS. Refer to the section **10.3.38 MMC_SetIsToLoadGlobalParams** for a detailed explanation.

```
void SetIsToLoadGlobalParams(
unsigned char ucVal
) throw (CMMCException)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

ucVal

The function recieves either 0 (not required to load the set global parameters) or 1 (required to load the set global parameters). +ve integer value

Return

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.6.10. SetHeartBeatConsumer

Sets the consumer heartbeat as an event to the user. Refer to the section **11.5.21 MMC_SetHeartBeatConsumer** for a detailed explanation.

```
void SetHeartBeatConsumer(
unsigned int uiHeartbeatTimeFactor
) throw (CMMCException)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

uiHeartbeatTimeFactor

Heart beat time factor is a multiple of 1 ms. The calculation of the basic cycle time (predetermined in the Resource file), multiplied by this heartbeat time factor, and 1 ms, will set the Heartbeat time.

Values accepted are:

0, not in use

>0, any +ve value

Return

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.6.11. CallbackFunc

Defines the callback function to be an integer with +ve values.

```
int CallbackFunc(
unsigned char*, short, void*
)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

unsigned char, short, void**

Character, short, or void with a +ve value

15.6.12. RegisterEventCallback

Registers the event callback for specific type callbacks. Refer to the section **12.19 Asynchronous Events Callback** for a detailed explanation of callback events.

```
void RegisterEventCallback(
MMC_EVENT_ENUM eClbType,
void * pfClbk
)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

MMC_EVENT_ENUM eClbType

The parameter MMC_EVENT_ENUM is a specialist enumerator variable with values:

MMCPP_PDORCV, MMCPP_HBEAT	0
MMCPP_MOTIONENDED	1
MMCPP_EMCY	2
MMCPP_ASYNC_REPLY	3
MMCPP_HOME_ENDED	4
MMCPP_MODBUS_WRITE	5
MMCPP_TOUCH_PROBE_ENDED	6
MMCPP_NODE_ERROR	7
MMCPP_STOP_ON_LIMIT	8
MMCPP_TABLE_UNDERFLOW	9

The eClbType parameter is called according to the above enumerator, and has values:

PdoRcvEventCallback
HBeateEventCallback
MotionEndEventCallback
EmergencyEventCallback
HomeEndedEventCallback
ModbusWriteEventCallback
SysErrorEventCallback
AsyncReplyEventCallback
TouchProbeEndCallback
NodeErrorEventCallback
StopOnLimitEventCallback
TableUnderflowEventCallback

The enumerator value must complement the callback type. Refer to the table in section **15.6.1 Event Type Definitions** for details of the Event type definitions.

pfClbk

Points to the callback function with no value returned.

15.6.13. RegisterSyncTimerFunction

```
void RegisterSyncTimerFunction(
MMC_SYNC_TIMER_CB_FUNC func,
unsigned short usSYNCTimerTime
)
```

Source GMAS\includes\CPP\MMCConnection.h

.NET Definition

Function Parameters

MMC_SYNC_TIMER_CB_FUNC func

[IN] Points to the callback function MMC_SYNC_TIMER_CB_FUNC using MMC_CreateSYNCTimer.

usSYNCTimerTime

[IN] Defines the time between which a synchronization message is sent as an event.

15.7. The CMMCNetwork class

The class MMCNetwork wraps the network communication functions detailed in the section **11.1 Network Function Blocks on page 11-665**. The diagram **in Figure 15-12** describes the heirarchial structure of the classes and type definitions associated with the MMCNetwork.

Figure 15-12 MMCNetwork class diagram

The class MMCNetwork retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-13 Fields and methods of the MMCNetwork class

The detailed class view shown in **Figure 15-13** describes the fields and methods associated with the MMCNetwork class. It should be noted that Protected and Private and Protected functions together with their operations, should be transparent to the user, and are not for general application by the user.

15.7.1. GetCommDiagnostics ResetCommDiagnostics

Refer to the section **11.7.10 MMC_GetEthercatCommStatistics** and **11.7.14 MMC_ResetCommDiagnostics on page 11-855 - 11-874** for details of the description, scope, and communication mode.

```
void GetCommDiagnostics(
MMC_GETCOMMDIAGNOSTICS_OUT& stOutParams
) throw (CMMCException);
```

```
void ResetCommDiagnostics(
MMC_RESETCOMMDIAGNOSTICS_OUT& stOutParams
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCNetwork.h

.NET Definition

Function Parameters

stOutParams

Defined output parameters function of MMC_GETCOMMDIAGNOSTICS_OUT, and MMC_RESETCOMMDIAGNOSTICS_OUT respectively.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name
Axis reference Error ID
Status of the axis.

15.7.2. ResetCommStatistics

Refer to the section **11.7.15 MMC_ResetCommStatistics on page 11-877** for details of the description, scope, and communication mode.

```
void ResetCommStatistics(
MMC_RESETCOMMSTATISTICS_OUT& stOutParams
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCNetwork.h

.NET Definition

Function Parameters

stOutParams

Defined output parameters function of MMC_RESETCOMMSTATISTICS_OUT.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name
Axis reference Error ID
Status of the axis.

15.7.3. GetNetworkInfo

Refer to the section **11.1.7 MMC_NetworkInfo on page 11-684** for details of the description, scope, and communication mode.

```
int GetNetworkInfo(
MMC_GETCOMMSTATISTICS_OUT& stOutParams
) throw (CMMCException);
```

```
int GetNetworkInfo(
MMC_NETWORKINFO_OUT& stOutParams
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCNetwork.h

.NET Definition

Function Parameters

stOutParams

Defined output parameters function of MMC_GETCOMMSTATISTICS_OUT and MMC_NETWORKINFO_OUT.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.8. The CMMCHostComm class

The class CMMCHostComm wraps the host communication functions detailed in the section **11.2 Host Communication on page 11-711**. The class CMMCHostComm retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-14 Fields and methods of the CMMCHostComm class

The detailed class view shown in **Figure 15-14** describes the fields and methods associated with the CMMCHostComm class. It should be noted that Private and Protected functions and their operation should be transparent to the user, and are not for general application by the user.

15.8.1. MbusStartServer

Refer to the section **11.3.5 MMC_MbusStartServer on page 11-724** for details of the description, scope, and communication mode.

```
void MbusStartServer(
unsigned int uiConnHndl,
unsigned short usID
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCHostComm.h

.NET Definition

Function Parameters

uiConnHndl

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

usID

Modbus start server enumerator ID has the following values:

MODBUS_NOT_STARTED = 0

MODBUS_RUNNING = 1

MODBUS_STOPPED = 2

After the G-MAS is powered-up, Modbus server is in the MODBUS_NOT_STARTED state – Initial state, the Modbus server does not exist.

After the Modbus server state is changed to MODBUS _RUNNING – the Modbus server is created, transmissions from Modbus clients will be handled by the server.

When the Modbus server state is changed to MODBUS_STOPPED – the Modbus server connection is removed, no transmissions will be handled from different Modbus clients.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.8.2. MbusStopServer

Refer to the section **11.3.6 MMC_MbusStopServer on page 11-727** for details of the description, scope, and communication mode.

```
void MbusStopServer(
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCHostComm.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.8.3. MbusReadHoldingRegisterTable

Refer to the section **11.3.3 MMC_MbusReadHoldingRegisterTable on page 11-718** for details of the description, scope, and communication mode.

```
void MbusReadHoldingRegisterTable(
int startRef,
int refCnt,
MMC_MODBUSREADHOLDINGREGISTERSTABLE_OUT& stOutParams
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCHostComm.h

.NET Definition

Function Parameters

startRef

Start Reference from the base coil table of linear parameters. Any +ve integer values accepted.

refCnt

Reference count. Any +ve integer values.

stOutParams

Refer to the **MMC_MODBUSREADHOLDINGREGISTERSTABLE_OUT Structure on page 11-719.**

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.8.4. MbusWriteHoldingRegisterTable

Refer to the section **11.3.8 MMC_MbusWriteHoldingRegisterTable on page 11-733** for details of the description, scope, and communication mode.

```
void MbusWriteHoldingRegisterTable(
MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN& stInParams
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCHostComm.h

.NET Definition

Function Parameters

stInParams

Refer to the **MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN Structure on page 11-734.**

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.8.5. MbusIsRunning

Refer to the section **11.3.1 MMC_MbusIsRunning on page 11-712** for details of the description, scope, and communication mode.

```
bool MbusIsRunning(
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCHostComm.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.8.6. MbusReadCoilsTable

Refer to the section **11.3.2 MMC_MbusReadCoilsTable on page 11-715** for details of the description, scope, and communication mode.

```
void MbusReadCoilsTable(
int startRef,
int refCnt,
MMC_MODBUSREADCOILS_OUT& stOutParams
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCHostComm.h

.NET Definition

Function Parameters

startRef

Start Reference from the base coil table of linear parameters. Any +ve integer values accepted.

refCnt

Reference count. Any +ve integer values.

stOutParams

Refer to the **MMC_MODBUSREADCOILS_OUT Structure on page 11-716.**

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.8.7. MbusWriteCoilsTable

Refer to the section **11.3.7 MMC_MbusWriteCoilsTable on page 11-730** for details of the description, scope, and communication mode.

```
void MbusWriteCoilsTable(
MMC_MODBUSWRITECOILS_IN& stInParams
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCHostComm.h

.NET Definition

Function Parameters

stInParams

Refer to the **MMC_MODBUSWRITECOILS_IN Structure on page 11-731.**

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.8.8. MbusReadInputsTable

Refer to the section **11.3.4 MMC_MbusReadInputsTable on page 11-721** for details of the description, scope, and communication mode.

```
void MbusReadInputsTable(
int startRef,
int refCnt,
MMC_MODBUSREADINPUTS_OUT& stOutParams
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCHostComm.h

.NET Definition

Function Parameters

startRef

Start Reference from the base coil table of linear parameters. Any +ve integer values accepted.

refCnt

Reference count. Any +ve integer values.

stOutParams

Refer to the **MMC_MODBUSREADINPUTS_OUT Structure on page 11-722.**

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.8.9. SetModbus[LongSwapped][Short]

Refer to the section **11.3.4 MMC_MbusReadInputsTable on page 11-721** for details of the description, scope, and communication mode.

```
void SetModbus[LongSwapped][ Short][ Float][FloatSwapped](
MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN& stInParams,
int iOffset,
[iRefCount],
long lPos,[short sPos],[float fPos],
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCHostComm.h

.NET Definition

Function Parameters

iOffset

The is the start reference parameter **StartRef** defined in the **MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN Structure on page 11-734**.

iRefCount

The is the reference count parameter **refCnt** defined in the **MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN Structure on page 11-734**.

long lPos,[short sPos],[float fPos]

Long, short, or float value of the array parameter described in **MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN Structure on page 11-734.**

stInParams

Refer to the **MMC_MODBUSWRITEHOLDINGREGISTERSTABLE_IN Structure on page 11-734.**

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.9. The CMMCModbusBuffer class

The class CMMCModbusBuffer wraps the host communication functions detailed in the section **11.2 Host Communication on page 11-711**. The class CMMCModbusBuffer retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-15 Fields and methods of the CMMCModbusBuffer class

The detailed class view shown in **Figure 15-15** describes the fields and methods associated with the CMMCModbusBuffer class. These are generally default parameters, which can be operated using their default values. However if the user wishes to change the defaults, refer to the relevant parameter section in the manual.

It should be noted that Private and Protected functions and their operation should be transparent to the user, and are not for general application by the user.

15.10. The CMMCModbusSwapBuffer class

The class CMMCModbusSwapBuffer wraps the host communication functions detailed in the section **11.2 Host Communication on page 11-711**. The class CMMCModbusSwapBuffer retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-16 Fields and methods of the CMMCModbusSwapBuffer class

The detailed class view shown in **Figure 15-16** describes the fields and methods associated with the CMMCModbusSwapBuffer class. It should be noted that Private and Protected functions and their operation should be transparent to the user, and are not for general application by the user.

15.11. The CMMCNode class

The class CMMCNode wraps the host communication functions detailed in the section **11.2 Host Communication on page 11-711**. The class CMMCNode retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-17 Fields and methods of the CMMCNode class

The detailed class view shown in **Figure 15-18** describes the fields and methods associated with the CMMCNode class. It should be noted that Private and Protected functions and their operation should be transparent to the user, and are not for general application by the user.

15.11.1. Reset

Refer to the section **5.7.18 MMC_Reset on page 5-199** for details of the description, scope, and communication mode.

```
void Reset(
) throw(CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name
Axis reference Error ID
Status of the axis.

15.11.2. ReadStatus

Refer to the section **5.7.17 MMC_ReadStatus on page 5-196** for details of the description, and scope.

```
unsigned long ReadStatus(
unsigned short& usAxisErrorID,
unsigned short& usStatusWord
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

usAxisErrorID

Returns the axis error bitwise ID defined by the following enumerators. Bitwise ID error code:

ID	Enumerator
0x1	MMC_ERR_TYPE_FAULT_BIT
0x2	MMC_ERR_TYPE_HEARTBEAT
0x4	MMC_ERR_TYPE_EMERGENCY
0x8	MMC_ERR_TYPE_COMM
0x10	MMC_ERR_TYPE_CFG_FILE

usStatusWord

Drive Status text. Any text characters.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.11.3. SendSDO

Refer to the section **11.7.16 MMC_SendSDO on page 11-880** for details of the description, and scope.

```
void SendSdoCmd(
long lData,
unsigned char ucService,
unsigned char ucSubIndex,
unsigned long ulDataLength,
unsigned short usIndex,
unsigned short usSlaveID
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

lData

Data. Unlimited +ve or –ve values (long).

ucService

Defines whether the input is a download or upload. Accepted integer values of:

1 - Download

2 - Upload

ucSubIndex

Defines which index value signifies the group of events to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. From events group 7 and above, the values represent the RPDO output. This parameter should mirror the enumerator value of the ucEventGroup variable, where applicable.

Any +ve character values.

ulDataLength

Length of the data. Unlimited +ve values in bytes.

usSlaveID

The slave ID. Any +ve integer value.

usIndex

COB index. Any +ve integer values.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.11.4. SendSDODownload

Refer to the section **11.7.16 MMC_SendSDO on page 11-880** for details of the description, and scope.

```
void SendSdoDownload(
long lData,
unsigned char ucSubIndex,
unsigned long ulDataLength,
unsigned short usIndex,
unsigned short usSlaveID
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

lData

Data. Unlimited +ve or –ve values (long).

ucSubIndex

Defines which index value signifies the group of events to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. From events group 7 and above, the values represent the RPDO output. This parameter should mirror the enumerator value of the ucEventGroup variable, where applicable.

Any +ve character values.

ulDataLength

Length of the data. Unlimited +ve values in bytes.

usSlaveID

The slave ID. Any +ve integer value.

usIndex

COB index. Any +ve integer values.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.11.5. SendSDOUpload

Refer to the section **11.7.16 MMC_SendSDO on page 11-880** for details of the description, and scope.

```
long SendSdoUpload(
unsigned char ucSubIndex,
unsigned long ulDataLength,
unsigned short usIndex,
unsigned short usSlaveID
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

ucSubIndex

Defines which index value signifies the group of events to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. From events group 7 and above, the values represent the RPDO output. This parameter should mirror the enumerator value of the ucEventGroup variable, where applicable.

Any +ve character values.

ulDataLength

Length of the data. Unlimited +ve values in bytes.

usSlaveID

The slave ID. Any +ve integer value.

usIndex

COB index. Any +ve integer values.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.11.6. SendSDOUploadAsync

Refer to the section **11.7.16 MMC_SendSDO on page 11-880** for details of the description, and scope.

```
void SendSdoUploadAsync(
unsigned char ucSubIndex,
unsigned long ulDataLength,
unsigned short usIndex,
unsigned short usSlaveID
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

ucSubIndex

Defines which index value signifies the group of events to be transferred from the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. From events group 7 and above, the values represent the RPDO output. This parameter should mirror the enumerator value of the ucEventGroup variable, where applicable.

Any +ve character values.

ulDataLength

Length of the data. Unlimited +ve values in bytes.

usSlaveID

The slave ID. Any +ve integer value.

usIndex

COB index. Any +ve integer values.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.11.7. RetreiveSDOUploadAsync

Refer to the section **11.7.16 MMC_SendSDO on page 11-880** for details of the description, and scope.

```
void RetreiveSdoUploadAsync(
long& lData
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

lData

Data. Unlimited +ve or –ve values (long).

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.11.8. PDOGeneralRead

Refer to the section **11.5.16 MMC_PDOGeneralRead on page 11-793** for details of the description, and scope.

```
MMCPPULL_T PDOGeneralRead(
unsigned char ucParam
) throw(CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

ucParam

Defines which of the general group of 32bit assigned events 16 or 17 is to be transferred to the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. Use the values 0, 1 denoted below to represent the assignment:

NC_COMM_EVENT_GROUP16 0

NC_COMM_EVENT_GROUP17 1

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.11.9. PDOGeneralWrite

Refer to the section **11.5.17 MMC_PDOGeneralWrite on page 11-796** for details of the description, and scope.

```
void PDOGeneralWrite(
unsigned char ucParam,
MMCPPULL_T ulliVal
) throw(CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

ulliVal

Use the *ulliVal* parameter to describe the value of the general PDO parameter. Values are +ve 64bit (8 bytes) character and/or integer.

ucParam

Defines which of the general group of 32bit assigned events 16 or 17 is to be transferred to the G-MAS. Refer to the section **11.4.3 PDO Mapping on page 11-738**. Use the values 0, 1 denoted below to represent the assignment:

NC_COMM_EVENT_GROUP16 0

NC_COMM_EVENT_GROUP17 1

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.11.10. GetPDOInfo

Refer to the section **11.5.14 MMC_GetPDOInfo on page 11-787** for details of the description, and scope.

```
void GetPDOInfo(
unsigned char uiPDONumber,
int &iPDOEventMode,
unsigned char &ucPDOCommType,
unsigned char &ucTPDOCommEventGroup,
unsigned char &ucRPDOCommEventGroup
) throw(CMMCException);
```

Source GMAS\includes\CPP\CMMCNode.h

.NET Definition

Function Parameters

uiPDONumber

The PDO index number. Any +ve character acceptable..

&iPDOEventMode

The PDO event mode integer

&ucPDOCommType

PDO communication type as a +ve character value

&ucTPDOCommEventGroup

TPDO communication event group as a +ve character value

&ucRPDOCommEventGroup

RPDO communication event group as a +ve character value

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID

Status of the axis.

15.12. The CMMCBulkRead class

The class CMMCBulkRead wraps the bulk parameters reading functions detailed in **Chapter 8: Bulk Parameters Reading on page 8-488**. The class CMMCBulkRead retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-18 Fields and methods of the CMMCBulkRead class

The detailed class view shown in **Figure 15-19** describes the fields and methods associated with the CMMCBulkRead class. It should be noted that Private and Protected functions and their operation should be transparent to the user, and are not for general application by the user.

15.12.1. CMMCBulkRead

Refer to the section **8.1.1 above MMC_ConfigBulkRead 8-489** for details of the description, and scope.

```
CMMCBulkRead(
MMC_CONNECT_HNDL hConnectHandle,
NC_BULKREAD_CONFIG_ENUM eConfig,
NC_BULKREAD_PRESET_ENUM ePreset
);
```

Source GMAS\includes\CPP\CMMCBulkRead.h

.NET Definition

Function Parameters

hConnectHandle

Connection handle

eConfig

Configuration enumerator defined by the following:

BULKREAD_DEFAULT_NUMBER_OF_AXES	2
BULKREAD_DEFAULT_FIRST_AXIS_REF	0
BULKREAD_DEFAULT_SECOND_AXIS_REF	1
BULKREAD_DEFAULT_IS_PRESET	1
BULKREAD_DEFAULT_PRESET	eNC_BULKREAD_PRESET_1
BULKREAD_DEFAULT_CONFIG	eBULKREAD_CONFIG_2

ePreset

Preset enumerator defined by the following options:

NC_BULKREAD_PRESET_1 m_pPreset1Data[NC_MAX_AXES_PER_BULK_READ];

NC_BULKREAD_PRESET_2 m_pPreset2Data;

NC_BULKREAD_PRESET_3 m_pPreset3Data[NC_MAX_AXES_PER_BULK_READ];

For further details, refer to **Chapter 8: Bulk Parameters Reading on page 8-488**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.12.2. Config

Refer to the section **8.1.1 above MMC_ConfigBulkRead 8-489** for details of the description, and scope.

```
void Config(
MMC_CONFIGBULKREAD_IN stCfgBulkReadIn
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCBulkRead.h

.NET Definition

Function Parameters

stCfgBulkReadIn

Refer to the section **MMC_CONFIGBULKREAD_IN Structure on page 8-490** for details of this parameter.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.12.3. BulkRead

Refer to the section **8.1.1 above MMC_ConfigBulkRead 8-489** for details of the description, and scope.

```
void BulkRead(
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCBulkRead.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.12.4. CMMCBulkRead Source Code Examples

```
MMC_CONFIGBULKREAD_IN stCfgIn;

stCfgIn.eConfiguration = eBULKREAD_CONFIG_2;
stCfgIn.uBulkReadParams.eBulkReadPreset = eNC_BULKREAD_PRESET_2;
stCfgIn.ucIsPreset = 1;
stCfgIn.usAxisRefArray[0] = 0;
stCfgIn.usAxisRefArray[0] = 1;

stCfgIn.usNumberOfAxes = 2;

CMMCBulkRead br(g_hConnect);
try
{
      br.SetDefaultParameters(stCfgIn);
      br.BulkRead();
}

catch (CMMCException e)
{
      cout << e.what() << endl;
}
```

Alternatively:

```
stCfgIn.eConfiguration = eBULKREAD_CONFIG_2;
stCfgIn.uBulkReadParams.eBulkReadPreset = eNC_BULKREAD_PRESET_2;
stCfgIn.ucIsPreset = 1;
stCfgIn.usAxisRefArray[0] = 0;
stCfgIn.usAxisRefArray[0] = 1;

stCfgIn.usNumberOfAxes = 2;

CMMCBulkRead br(g_hConnect);
try
{
      br.Config(stCfgIn);
      br.BulkRead();
}

catch (CMMCException e)
{
      cout << e.what() << endl;
}
```

15.13. The CMMCMotionAxis class

The base class for PVT (ECAM and splines, in future) is the class CMMCMotionAxis, which is inherited from CMMCAxis, i.e. it contains all CMMCAxis members and methods. The class CMMCMotionAxis functions are detailed in **Chapter 9: Position, Velocity, Time (PVT) Motion**. The class CMMCMotionAxis retains the same field parameter properties and values described in this document for the C function blocks.

Figure 15-19 Fields and methods of the CMMCMotionAxis class

The detailed class view shown in **Figure 15-20** describes the fields and methods associated with the CMMCMotionAxis class.

It should be noted that Private and Protected functions and their operation should be transparent to the user, and are not for general application by the user.

The base class for PVT (ECAM and splines, in future) called CMMCMotionAxis is inherited from CMMCAxis, i.e. it contains all CMMCAxis members and methods. The methods for CMMCMotionAxis are:

virtual void BindAxis(CMMCAxis& pAxis) **throw** (CMMCException);

Since the class is not axis-related, prior to using it, the user should decide the axis on which PVT is to be performed. The above method may then be called prior to using any PVT (ECAM) methods.

inline virtual void InitAxisData(**char*** cName, MMC_CONNECT_HNDL hConn) **throw** (CMMCException)

This method is inherited from CMMCAxis class, but overloaded in the CMMCMotionAxis class, since the method does not suit to the requirements of the CMMCAxis implementation. The BindAxis method is used instead, and InitAxisData will just throw an exception.

15.14. The CMMCPVT class

The class CMMCPVT wraps the motion axis parameter reading functions detailed in **Chapter 9: Position, Velocity, Time (PVT) Motion**. The class CMMCPVT retains similar field parameter properties and values described in this document for the C function blocks.

Figure 15-20 Fields and methods of the CMMCPVT class

The detailed class view shown in **Figure 15-21** describes the fields and methods associated with the CMMCMotionAxis class. It should be noted that Private and Protected functions and their operation should be transparent to the user, and are not for general application by the user.

The CMMCPVT class is inherited from the CMMCMotionAxis. It contains the following methods:

Function	Explanation
InitPVTTable	The method is a wrapper for the MMC_InitTableCmd() C command. InitTableCmd().
LoadPVTTable	The method loads PVT table from file
AppendPointsToPVTTable	This method appends points to the current PVT table (in automatic mode).
AppendPointsToPVTTable	This method appends points to the current PVT table (in manual mode).
MovePVT	This method inserts PVT function block.
UnloadPVTTable	Unloads PVT table

15.14.1. InitPVTTable

Initiates the PVT Table. Refer to the section **9.7.1 MMC_InitTable** for details of the description, and scope.

```
void InitPVTTable(
unsigned long ulMaxPoints,
unsigned long ulUnderflowThreshold,
unsigned char ucIsCyclic,
unsigned char ucIsDynamic,
unsigned char ucIsPosAbsolute
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCPVT.h

.NET Definition

Function Parameters

ulMaxPoints

The maximal number of points that the table will be able to contain (in non-cyclic mode). Any +ve values accepted.

ulUnderflowThreshold

An event will be generated if the number of points between the current index, and the end index falls below this value. Value cannot be greater that the ulMaxPoints value. Any +ve values accepted.

ucIsCyclic

Is the table supposed to be cyclic? i.e. when the index reaches the end of the table, will it roll over and start from the beginning. Boolean answer 0 or 1

ucIsDynamic

Is dynamic append allowed? Boolean answer 0 or 1

ucIsPosAbsolute

Is position absolute? Boolean answer 0 or 1

throw (CMMCException)

Refer to the **15.1.1 CMMCException**. Produces details of the error including; Function Name, Structure name, Axis reference, Error ID, Status of the axis

15.14.2. LoadPVTTable

This method loads PVT table from file. Refer to the section **9.7.2 MMC_LoadTableFromFile** for details of the description, and scope.

```
void LoadPVTTable(
char* pFileName
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCPVT.h

.NET Definition

Function Parameters

pFileName

The file name in standard UNIX format. String value (null-terminated).

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including; Function Name, Structure name, Axis reference, Error ID, Status of the axis

15.14.3. AppendPointsToPVTTable

The method appends points to the current PVT table (in automatic and manual modes). Refer to the section **9.7.5 MMC_AppendPointsToTable** for details of the description, and scope.

Append automatically

```
void AppendPointsToPVTTable(
double* dTable,
unsigned long ulNumberOfPoints,
unsigned char ucIsTimeAbsolute
) throw (CMMCException);
```

Append Manually

```
void AppendPointsToPVTTable(
double* dTable,
unsigned long ulNumberOfPoints,
unsigned long ulStartIndex,
unsigned char ucIsTimeAbsolute
) throw (CMMCException);
```

Source GMAS\includes\CPP\CMMCPVT.h

.NET Definition

Function Parameters

dTable

Pointer to the Table of values. Table of limited length in values, limited to 170 values max.

ulNumberOfPoints

Number of points to append. Any +ve values accepted.

ulStartIndex

Index to manually append from. Any +ve values accepted. Used only in manual mode.

ucIsTimeAbsolute

Boolean result to the question, Is the Time Absolute or not? Select the mode of the parameter:

0 - absolute mode
Every "time" input is used as absolute time when the current point will end the motion and reach the desired position.

1 - relative mode
Every "time" input is used as the time that will take to move from previous point to the end of current point.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including; Function Name, Structure name, Axis reference, Error ID, Status of the axis

15.14.4. MovePVT

This method inserts a PVT function block. Refer to the section **9.7.4 MMC_MoveTable** for details of the description, and scope.

```
void MovePVT(
MC_COORD_SYSTEM_ENUM eCoordSystem = MC_NONE_COORD
);
```

Source GMAS\includes\CPP\CMMCPVT.h

.NET Definition

Function Parameters

eCoordSystem

Define the types of supported coordinate systems. The MC_COORD_SYSTEM_ENUM enumerator options are:

MC_NONE_COORD = 0 default value

MC_ACS_COORD = 1, Not supported at this time

MC_MCS_COORD = 2

MC_PCS_COORD = 3, Not supported at this time

15.14.5. UnloadPVTTable

This method unloads the PVT table from the G-MAS. Refer to the section **9.7.3 MMC_UnloadTable** for details of the description, and scope.

```
void UnloadPVTTable(
void
);
```

Source GMAS\includes\CPP\CMMCPVT.h

.NET Definition

Function Parameters

void

Function takes no parameters

15.14.6. CMMCPVT Source Code Examples

```
eTableType = eNC_TABLE_PVT_ARRAY;
ucIsCyclic = 1;
ucIsDynamicMode = 1;
ucIsPosAbsolute = 0;
ulMaxNumberOfPoints = 50;
ulUnderflowThreshold = 6;

usAxisRef = g_cGroupObject.GetRef();
usDimension = NUMBER_OF_AXES;

try
{
g_cPVTObject.BindAxis(g_cGroupObject);
      g_cPVTObject.InitPVTTable(ulMaxNumberOfPoints, ulUnderflowThreshold, ucIsCyclic,
ucIsDynamicMode, ucIsPosAbsolute);
g_cPVTObject.AppendPointsToPVTTable(stAppendIn.dTable, stAppendIn.ulNumberOfPoints, (unsigned
char )0);
g_cPVTObject.MovePVT(MC_ACS_COORD);
}

catch (CMMCException ex)
{
      cout << "Failed to init, error [" << ex.error() << "]" << endl;
}
```

Or, using file mode:

```
try
{
g_cPVTObject.BindAxis(g_cGroupObject);
g_cPVTObject.LoadPVTTable(pFileName);
g_cPVTObject.MovePVT(MC_ACS_COORD);
}

catch (CMMCException ex)
{
      cout << "Failed to init, error [" << ex.error() << "]" << endl;
}
```

15.15. The MMCEIPSession class

The class MMCEIPSession wraps the network communication functions detailed in the section . The diagram in **Figure 15-22** describes the heirarchial structure of the classes and type definitions associated with the MMCEIPSession.

The class MMCEIPSession retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-21 Fields and methods of the MMCEIPSession class

The detailed class view shown in **Figure 15-22** describes the fields and methods associated with the MMCEIPSession class. It should be noted that Protected and Private and Protected functions together with their operations, should be transparent to the user, and are not for general application by the user.

15.15.1. EIPCloseSession

Closes an EthernetIP session. Refer to the section **11.10.15 EIPCloseSession** for details of the description, scope, and communication mode.

```
int EIPCloseSession(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCEIPSession.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name Structure name
Axis reference Error ID
Status of the axis.

15.15.2. EipCreate

Creates a new EthernetIP session. Refer to the section **11.10.16 EipCreate** for details of the description, scope, and communication mode.

```
int EipCreate(
char *strPath
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCEIPSession.h

.NET Definition

Function Parameters

**strPath*

Path of the EtherNETIP session. Value to maximum of 80 characters. Refer to the similar parameter *cPath* in the section **11.10.16 EipCreate EIP_CREATE_IN Structure**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.15.3. EipDestroy

Kills the present EtherNETIP session. Refer to the section **11.10.17 EipDestroy** for details of the description, scope, and communication mode.

```
int EipDestroy(
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCEIPSession.h

.NET Definition

Function Parameters

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.15.4. EipOpenSession

Opens a new EIP session. Refer to the section **11.10.14 EipOpenSession** for details of the description, scope, and communication mode.

```
int EipOpenSession(
EIP_CALLBACK_FUNC pClbkFunc,
bool bEventNotification = true
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCEIPSession.h

.NET Definition

Function Parameters

pClbkFunc

The value of the EIP callback function which produces the appropriate event. Refer to the details on the callback function in section **12.16 Ethernet-IP Event**.

bEventNotification = true

Boolean answer to the question; Is the event notification true? 0 or 1

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.16. The CMMCEIPDataType class

The class CMMCEIPDataType wraps the network communication functions detailed in the section . The diagram in **Figure 15-22** describes the heirarchial structure of the classes and type definitions associated with the CMMCEIPDataType. These define the various EthernetIP data type variables and their possible values.

The class CMMCEIPDataType retains the same field parameter properties and values described in this document for the C function blocks, and while small visual changes may be made to some variables, these are transparent, and do not change the operation of the variable.

Figure 15-22 Fields and methods of the CMMCEIPDataType class

The detailed class view shown in **Figure 15-22** describes the fields and methods associated with the CMMCEIPDataType class. It should be noted that Protected and Private and Protected functions together with their operations, should be transparent to the user, and are not for general application by the user.

15.16.1. EipTagInit

Obtains a tag setting by name (*szName) or by instance (iInstance) and initializes this object with tag properties. Refer to the section **11.10.8 EipGetDevTagRefByName** and **11.10.9 EipSetDevTag** for details of the description, scope, and communication mode.

```
int EipTagInit(
const char *szName
[int iInstance]
) throw (CMMCException);
```

Source GMAS\includes\CPP\ CMMCEIPDataType.h

.NET Definition

Function Parameters

**szName*

Tag/assembly name as declared in XML configuration file. Refer to the parameter *cName* in section **11.10.8 EipGetDevTagRefByName EIP_REFBYNAME_IN Structure**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.16.2. EipSetTag

Writes data to adapter memory. Refer to the section **11.10.9 EipSetDevTag** for details of the description, scope, and communication mode.

```
int EipSetTag(
T tData[]
[T tData]
) throw (CMMCException);
```

Source GMAS\includes\CPP\ CMMCEIPDataType.h

.NET Definition

Function Parameters

T tData[],T tData

Retrieve data of the referenced adapter tag either as an array (buffer) or a single dta value. Refer to the section **11.10.9 EipSetDevTag EIP_SETDEVTAG_IN Structure**.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:
Function Name Structure name
Axis reference Error ID
Status of the axis.

15.16.3. EipGetTag

Obtains data from the adapter memory into given tData or tVar buffer. Refer to the section **11.10.10 EipGetDevTag** for details of the description, scope, and communication mode.

```
int EipGetTag(
T tData[],
[T& tVar]
bool bSync=false,
) throw (CMMCException);
```

Source GMAS\includes\CPP\ CMMCEIPDataType.h

.NET Definition

Function Parameters

T tData[]

Retrieve data of the referenced adapter tag either as an array (buffer) or a single dta value. Refer to the sections **11.10.9 EipSetDevTag EIP_SETDEVTAG_IN Structure**, and **11.10.10 EipGetDevTag EIP_GETDEVTAG_IN Structure.**

T& tVar

The variable reference retrieved value. This is a returned output.

bool bSync=false

bSync synchronous flag. The default is Asynchronous or false (EipGetDevTag), otherwise use EipSyncGetDevTag.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.16.4. EipCheckReply

Answers whether or not data for the last request has arrived. Refer to the section **11.10.16 EipCreate** for details of the description, scope, and communication mode.

```
int EipCheckReply(
short& sStatus
) throw (CMMCException);
FORCEDINLINE bool EipIsWaiting(
){return _iDevWaitStatus;}
```

Source GMAS\includes\CPP\ CMMCEIPDataType.h

.NET Definition

Function Parameters

sStatus

The status is returned as either 0 meaning OK, or 1, Wait.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

FORCEDINLINE bool EipIsWaiting

This answers the question whether the devise tag is in waiting state or not. The FORCEDINLINE creates a macro which returns the return _iDevWaitStatus

15.16.5. EipGetData

Obtain data which has arrived asynchronously from a request. Refer to the sections **11.10.8 EipGetDevTagRefByName**, **11.10.10 EipGetDevTag** for details of the description, scope, and communication mode.

```
int EipGetData(
T tData[]
[T& tVar]
) throw (CMMCException);
```

Source GMAS\includes\CPP\ CMMCEIPDataType.h

.NET Definition

Function Parameters

T tData[]

Retrieve data of referenced adapter tag dependant on the index reference and buffer of the tag. Refer to the section **11.10.9 EipSetDevTag EIP_SETDEVTAG_IN Structure**, and **11.10.10 EipGetDevTag EIP_GETDEVTAG_IN Structure.**

T& tVar

The variable reference retrieved value. This is a returned output.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.16.6. EthernetIP Source Code Examples

```
#include "EIP_API.h"
#include "MMCEIPDataType.h"
#include "MMCEIPSession.h"

#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <sys/time.h>

#define NO_CALLBACK_EVENTS          0
#define DO_CALLBACK_EVENTS          1
#define _RESOURCE_XML_FILE          "/mnt/jffs/usr/eippdemo.xml"
//////////////////////////////////////////////////////////////////////////
//                              Globals                                 //
//////////////////////////////////////////////////////////////////////////
int gTerminateFlag;

struct timeval tv1, tv2;
struct timezone tz;
long timeOffset = 0;

#define _EIPP 1 //one may want to test sinple C interface as well

#ifdef _EIPP
enum _DTYPE_ID {
     eARR_INDEX = 0,
     eSINGLE_INDEX
} DTYPE_ID;

/*
 * Please note type in use must comply with XML type definitions for each tag
 */
CMMCEIPDataType<char> g_devPlc2Gmas[2]; //simply store twos tags(single/multidimensional)
CMMCEIPDataType<char> g_devGmas2Plc[2];//simply store twos tags (single/multidimensional)
CMMCEIPDataType<short> g_adpPlc2Gmas;
CMMCEIPDataType<short> g_adpGmas2Plc;
CMMCEIPDataType<short> g_adpOne2Gmas;
CMMCEIPDataType<short> g_adpGmas2One;
CMMCEIPDataType<int>       g_asmPlc2Gmas;
CMMCEIPDataType<int>       g_asmGmas2Plc;

#else //not _EIPP
#endif //_EIPP

//////////////////////////////////////////////////////////////////////////
//                            Function Prototypes                       //
//////////////////////////////////////////////////////////////////////////
void terminate_application(int signum);

//////////////////////////////////////////////////////////////////////////
//                            External Functions                          //
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
//                            Implementation                              //
//////////////////////////////////////////////////////////////////////////

/*
 * general implementation of EtherNetIP call-back function.
 */
int fnCallback(unsigned char* ucBuffer, short sReqID, void* pSock)
{
     unsigned char ucEventID = ucBuffer[2];
     switch (ucEventID)
     {
```

```
        case NM_REQUEST_RESPONSE_RECEIVED:
//              printf("NM_REQUEST_RESPONSE_RECEIVED: sReqID = %d\n", sReqID);
         break;
         case NM_ASSEMBLY_NEW_INSTANCE_DATA:
//              printf("NM_ASSEMBLY_NEW_INSTANCE_DATA: assembly instance = %d\n", sReqID);
                break;
         case NM_ASSEMBLY_NEW_MEMBER_DATA:
                //New data received for the specified assembly member. sReqID contains assembly
instance.
                printf("NM_ASSEMBLY_NEW_MEMBER_DATA: assembly instance = %d\n", sReqID);
               break;
         case NM_REQUEST_FAILED_INVALID_NETWORK_PATH:
                break;
         case NM_REQUEST_TIMED_OUT:
         printf("NM_REQUEST_TIMED_OUT: sReqID = %d\n", sReqID);
                break;
         case NM_CONNECTION_ESTABLISHED:
//              printf("\033[1;45m\tEIPTestAll: New connection opened with instance %d\033[0m\n",
sReqID);
                break;
         case NM_CONNECTION_VERIFICATION:
                printf("NM_CONNECTION_VERIFICATION\n");
                break;
        case NM_CONNECTION_RECONFIGURED:
         printf("NM_CONNECTION_RECONFIGURED\n");
         break;
         case NM_CONNECTION_TIMED_OUT:
//              printf("\033[1;42m\tEIPTestAll: Connection with instance %d timed out\033[0m\n",
sReqID);
                break;
         case NM_CONNECTION_CLOSED:
//              printf("\033[1;43m\tEIPTestAll: Connection with instance %d closed\033[0m\n",
sReqID);
                break;
         case NM_CLIENT_OBJECT_REQUEST_RECEIVED:
                printf("NM_CLIENT_OBJECT_REQUEST_RECEIVED\n");
                break;
         case NM_PENDING_REQUESTS_LIMIT_REACHED:
                printf("NM_PENDING_REQUESTS_LIMIT_REACHED\n");
                break;

         default:
                printf("%s Unhandled(unknown) response event. %d\n", __func__, ucEventID);
                break;
     }
     return 0;
}

/*
 * general initialization for these kind of applications
 */
int MainInit() {
     long time;
     struct sigaction stSigAction;

     //whenever a signal is caught, call terminate_application function
     stSigAction.sa_handler = terminate_application;

     sigaction(SIGINT, &stSigAction, NULL);
     sigaction(SIGTERM, &stSigAction, NULL);
     sigaction(SIGABRT, &stSigAction, NULL);
     sigaction(SIGQUIT, &stSigAction, NULL);

     /* Initialise peformance counter */
     /* ----------------------------- */
     for (int i = 0; i < 100; i++)
     {
         gettimeofday(&tv1, &tz);
         usleep(1000);
```

```
        gettimeofday(&tv2, &tz);
        time = (tv2.tv_sec - tv1.tv_sec)*1000000 + (tv2.tv_usec - tv1.tv_usec);
        timeOffset += (1000 - time);
    }
    timeOffset /= 100;

    return 0;
}

#ifndef _EIPP
#else //_EIPP

/*
 * Sample for EtherNetIp opening session via CPP interface
 */
int OpenSessionPP() {
    printf("Opening CPP EIP session\n");
    CMMCEIPSession::Instance()->EipOpenSession(fnCallback, false);
    CMMCEIPSession::Instance()->EipCreate(_RESOURCE_XML_FILE);
    return 0;
}
/*
 * Sample for EtherNetIp closing session via CPP interface
 */
int CloseSessionPP() {
    printf("Free EIP session memory\n");

    CMMCEIPSession::Instance()->EipDestroy();

    printf("Closing CPP EIP session\n");

    CMMCEIPSession::Instance()->EIPCloseSession();
    return 0;
}

#endif //_EIPP

/*
 *  Init device tags reference
 */
#ifndef _EIPP
#else //_EIPP

/*
 * Sample for adapter tags operations
 */
int AdpTESTPP() {
    int iHBCount;
    short iBuffer[7];
    static short iVar=0;

    g_adpPlc2Gmas.EipGetTag(iBuffer);
    g_adpOne2Gmas.EipGetTag(iVar);
    iHBCount = (int)iBuffer[0];
    //set heartbit
    iBuffer[0] =  (iHBCount > 0x7FFFFFFE) ? 0 : iHBCount + 1; //limit to max signed integer
0x7FFFFFFF.
    g_adpGmas2Plc.EipSetTag(iBuffer);
    g_adpGmas2One.EipSetTag(iVar+1);

    return 0;
}
/*
 * Sample for adapter tags initialization. Must be invoked before any other operation on tags.
 */
int AdpInitPP() {
    int rc = g_adpPlc2Gmas.EipTagInit("adp_plc2gmas");
    if (!rc) {
```

```
        rc  = g_adpGmas2Plc.EipTagInit("adp_gmas2plc");
        if (!rc) {
            rc  = g_adpOne2Gmas.EipTagInit("adp_one2gmas");
            if (!rc) {
                rc  = g_adpGmas2One.EipTagInit("adp_gmas2one");
            }
        }
    }
    return rc;
}

/*
 * Sample for device tags operations
 */
int DevTESTPP() {
    char  iBuffer[7];
    char iVar;
    short sReplyStatus;
    //
    //read device tag asynchronous
    //

    //single example
    if(!g_devPlc2Gmas[eSINGLE_INDEX].EipIsWaiting())
    {
        g_devPlc2Gmas[eSINGLE_INDEX].EipGetTag(NULL);
    }
    else
    {
        g_devPlc2Gmas[eSINGLE_INDEX].EipCheckReply(sReplyStatus);
        if (sReplyStatus == EIPP_RESPONSE_OK)
        {
            g_devPlc2Gmas[eSINGLE_INDEX].EipGetData(iVar);
            g_devGmas2Plc[eSINGLE_INDEX].EipSetTag(iVar+1);//heartbeat
        }
    }

    //multidimensional example
    if(!g_devPlc2Gmas[eARR_INDEX].EipIsWaiting())
    {
        g_devPlc2Gmas[eARR_INDEX].EipGetTag(NULL);
    }
    else
    {
        g_devPlc2Gmas[eARR_INDEX].EipCheckReply(sReplyStatus);
        if (sReplyStatus == EIPP_RESPONSE_OK)
        {
            g_devPlc2Gmas[eARR_INDEX].EipGetData(iBuffer);
            iBuffer[0]++;// heartbeat
            g_devGmas2Plc[eARR_INDEX].EipSetTag(iBuffer);
        }
    }
    return 0;
}

/*
 * Sample for device tags initialization. Must be invoked before any other operation on tags.
 */
int DevInitPP() {
    int rc = g_devPlc2Gmas[eARR_INDEX].EipTagInit("dev_plc2gmas");

    if (!rc) {
        rc  = g_devGmas2Plc[eARR_INDEX].EipTagInit("dev_gmas2plc");
        if (!rc) {
            rc  = g_devPlc2Gmas[eSINGLE_INDEX].EipTagInit("dev_one2gmas");
            if (!rc) {
                rc  = g_devGmas2Plc[eSINGLE_INDEX].EipTagInit("dev_gmas2one");
            }
```

```
        }
    }
    if(rc)
        printf("DevInitPP Failed\n");
    return rc;
}

/*
 * Sample for Assembly tags initialization. Must be invoked before any other operation on
tags.
 */
int AsmInitPP()
{
    int rc = g_asmPlc2Gmas.EipTagInit("asm_plc2gmas"); //initialization by name
    /*or by instances
     * int rc = g_asmPlc2Gmas.EipTagInit(2); //initialization by instance (must comply to XML
definitions)
     * */
    if (!rc) {
        rc = g_asmGmas2Plc.EipTagInit("asm_gmas2plc");
        /*or by instances
         * int rc = g_asmPlc2Gmas.EipTagInit(1); //initialization by instance (must comply to
XML definitions)
         * */
    }
    return rc;
}

/*
 * Sample for Assembly tags operations
 */
int AsmTESTPP()
{
    static int prev_data = 0;
    static int in_data;
    int iBuffer[7]; //type and dimension must comply to tyep and size definitions in XML

    g_asmPlc2Gmas.EipGetTag(iBuffer);
    in_data = iBuffer[0];

    //set changes only if first cell is different (this is only an application preference)
//  if (prev_data != in_data) {
        prev_data = in_data;
        iBuffer[0] = in_data + 1;
        g_asmGmas2Plc.EipSetTag(iBuffer);
//  }
    return 0;
}

#endif //_EIPP

/*
 * just displays a propeller for progress notification.
 */
void DisplayProgress() {
    static int iProgressIndex = 0;
    static char *cClock[] = {"-", "\\", "|", "/",  "-", "\\", "|", "/"};
    fprintf(stderr, "%s\r", cClock[iProgressIndex]);
    if (++iProgressIndex>7) iProgressIndex = 0;
}

/*****************************************************************************
 * FUNCTION:            terminate_application
 * DESCRIPTION:
 * INPUTS:              N/A
 * OUTPUTS:
 * RETURN VALUE:
 *****************************************************************************/
void terminate_application(int signum)
```

```
{
     gTerminateFlag = 1;
}

/*****************************************************************************
 * FUNCTION:              Initialize
 * DESCRIPTION:
 * INPUTS:                N/A
 * OUTPUTS:
 * RETURN VALUE:
 *****************************************************************************/
int main(int argc, char** argv)
{
     static int iClock = 0;

     MainInit();

#ifdef _EIPP
     OpenSessionPP();
     //initializations
     if (AdpInitPP())
     {
          printf("Addpters initialization faild\n" );
          exit (1);
     }
     if (DevInitPP())
     {
          printf("Addpters initialization faild\n" );
          exit (1);
     }

     if (AsmInitPP())
     {
          printf("Assemblies initialization faild\n" );
          exit (1);
     }

     while (!gTerminateFlag)
     {
          usleep(5000); //5 ms

          /* Device tags TEST */
          /* ---------------- */
          DevTESTPP();

          /* Adapter tags TEST */
          /* ----------------- */
          AdpTESTPP();

          /* assemblies TEST */
          /* --------------- */
          AsmTESTPP();
          if (!(++iClock%200))
               DisplayProgress();
//        if (iClock > 400)
//             break;
     }

     CloseSessionPP();

#else // not _EIPPP
#endif //_EIPPP
     return 0;
}
```

15.17. TCP/IP and UDP C++ User 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 definition of the socket port is managed by the Class using a simple interface for the following commands:

- **int Create** (unsigned short usPort, SOCK_CLBK fnClbk=NULL, int iMsgMaxSize=512);
- **int Receive** (void * pData, unsigned short usSize, long lDelay=0L, sockaddr_in* pSockaddr=NULL);
- **int Send** (void * pData, unsigned short usSize, sockaddr_in* pSockaddr=NULL);
- **int Connect** (char* szAddr, unsigned short usPort, bool& bWait);

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

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)

15.18. The CMMCUDP class

The class CMMCUDP wraps the host communication functions detailed in the section **11.2 Host Communication on page 11-711**. The EoE class is derived from the CMMCUDP class and uses EoE communication over UDP. This section describes the public interface for EoE.

Figure 15-23 Fields and methods of the CMMCUDP class

The detailed class view shown in **Figure 15-17** describes the fields and methods associated with the CMMCUDP class. It should be noted that Private and Protected functions and their operation should be transparent to the user, and are not for general application by the user.

15.18.1. Create

Create a connection to the G-MAS, given the host IP address, and Port number.

```
int Create (
char * cIP,
int iPort,
int iGMASPort = 0
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

cIP

IP address. Char accepted in xx:xx:xx:xx format

iPort

Port connection used. Integers accepted

iGMASPort = 0

G-MAS Port reverted to 0.

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.18.2. SendTo

Sends a message to the G-MAS.

```
int SendTo(
char* msg,
short sLength
) throw (CMMCException);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

msg

Message sent. Char accepted

sLength

Length of the message. String values

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.18.3. ReceiveFrom

Recieves a message from the G-MAS.

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

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

msg

Message sent. Char accepted

sLength

Length of the message. 2 byte ±ve value

sTimeout

Length of timeout. 2 byte ±ve value

throw (CMMCException)

Refer to the section **15.1.1 CMMCException**. Produces details of the error including:

Function Name	Structure name
Axis reference	Error ID
Status of the axis.	

15.18.4. SetTimeout

Sets the default time out for the UDP socket.

```
void SetTimeout(
long lTimeOut
) { this->_lTimeOut = lTimeOut;}
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

lTimeOut

The default time out for the UDP socket. Any value is acceptable.

15.18.5. ElmoSetAsyncArray

Send an EoE array command asynchronously, and sets parameter of type int or float. Refer to the section **11.8.8 MMC_ElmoSetArray** for details of the description, and scope.

```
int ElmoSetAsyncArray(
char cCmd[3],
short iIndex,
[const int iVal]
[const float fVal]
);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iIndex

EoE array index. Array index parameter (only relevant for array situations). Any +ve integer values.

iVal, fVal

Integer value of the data that is to be set. Optionally, float value of the data.

RETURN Returns 1 if command was sent OK, or alternative obtained the real value, otherwise 0.

15.18.6. ElmoSetAsyncParameter

Send an EoE command asynchronously, and sets parameter of type int or float. Refer to the section **11.8.3 MMC_ElmoSetParameter** for details of the description, and scope.

```
int ElmoSetAsyncParameter(
char cCmd[3],
[const int iVal]
[const float fVal]
);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iVal, fVal

Integer value of the data that is to be set. Optionally, float value of the data.

RETURN Returns 1 if obtained the integer or alternative real value, otherwise 0.

15.18.7. ElmoSetArray

Send an EoE array command, and sets parameter of type int or float. Refer to the section **11.8.8 MMC_ElmoSetArray** for details of the description, and scope.

```
int ElmoSetArray(
char cCmd[3],
short iIndex,
[const int iVal]
[const float fVal]
);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iIndex

EoE array index. Array index parameter (only relevant for array situations). Any +ve integer values.

iVal, fVal

Integer value of the data that is to be set. Optionally, float value of the data.

RETURN Return - 0 on success, otherwise 1.

15.18.8. ElmoSetParameter

Set an EoE command, and sets parameter of type int or float. Refer to the section **11.8.3 MMC_ElmoSetParameter** for details of the description, and scope.

```
int ElmoSetParameter(
char cCmd[3],
[const int iVal]
[const float fVal]
);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iVal, fVal

Integer value of the data that is to be set. Optionally, float value of the data.

RETURN Return - 0 on success, otherwise 1.

15.18.9. ElmoGetAsyncArray

Sends an asynchronous array command to get EoE parameter, with the result that one may get the following result asynchronously; Call IsReady, then ElmoReadData API. Refer to the section **11.8.5 MMC_ElmoGetArray** for details of the description, and scope.

```
int ElmoGetAsyncArray(
char cCmd[3],
[short iIndex]
);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iIndex

EoE array index. Array index parameter (only relevant for array situations). Any +ve integer values.

RETURN Return - 0 on success, otherwise 1.

15.18.10. ElmoGetAsyncParameter

Sends an asynchronous command to get EoE parameter, with the result that one may get the following result asynchronously; Call IsReady, then ElmoReadData API. Refer to the section **11.8.4 MMC_ElmoGetParameter** for details of the description, and scope.

```
int ElmoGetAsyncArray(
char cCmd[3],
);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

RETURN Return - 0 on success, otherwise 1.

15.18.11. ElmoGetArray

Sends an array command to get EoE parameter of type int or float, with the result that one may get the following result; Call IsReady, then ElmoReadData API. Refer to the section **11.8.5 MMC_ElmoGetArray** for details of the description, and scope.

```
int ElmoGetArray(char cCmd[3], short iIndex);
char cCmd[3],
short iIndex
[int& iVal]
[float& fiVal]
);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

cCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

iIndex

EoE array index. Array index parameter (only relevant for array situations). Any +ve integer values.

Int& iVal, float& fVal

Integer value of the data that is to be set. Optionally, float value of the data.

RETURN Return - 0 on success, otherwise 1.

15.18.12. ElmoGetParameter

Obtain EoE parameter of type int, float, or string. Refer to the section **11.8.4 MMC_ElmoGetParameter** for details of the description, and scope.

```
int ElmoGetParameter(
char szCmd[3],
[int& iVal,]
[float& fiVal,]
[char szVal[],]
[int iSize]
);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

szCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

Int& iVal, float& fVal, or szVal[]

Integer value of the data that is to be set. Optionally, float or string value of the data.

iSize

Storage buffer for the returned value.

RETURN Return - 0 on success, otherwise 1.

15.18.13. ElmoReadData

Read EoE data and set iValue accordingly parameter of type int, float, or string. Refer to the section **11.8.5 MMC_ElmoGetArray** for details of the description, and scope.

```
int ElmoReadData(
[char* cBuffer,]
[int iSize]
[int& iValue,]
[float& fiVal]
);
```

Source GMAS\includes\CPP\MMCUDP.h

.NET Definition

Function Parameters

szCmd[3]

Name of the parameter limited to three characters. Any +ve character value with a maximum of 2 bytes

Int& iVal, float& fVal

Integer value of the data that is to be set. Optionally, float value of the data.

iSize

Storage buffer for the returned value.

RETURN Return - 0 on success, otherwise 1.

Chapter 16: Appendix

16.1. Axis Parameters (Explanations)

The axis parameters define the MMC_PARAMETER_LIST_ENUM eParameterNumber values of the axis status. Refer to the section **5.3.2 Parameters Tables** for the integer parameter definitions.

Table 7: Boolean Global Parameters table

Index	Parameter description	Explanation
6	MOVE_TYPE	
8	CTRL_MODE	
9	REAL_CTRL_MODE	
29	MMC_CYCLE_TIME_PARAM	Cycle time of the G-MAS
30	MMC_RES_ID_PARAM	Resource file ID, created by EAS to identify the resource file.
49	MMC_CONNECTION_TYPE_PARAM	
53	MMC_SUPPORT_BLENDED_FB_PENDING	
54	FB_CNT_BEFORE_ACTIVE_FB_ON_END_VELOCITIES_RECALCULATION	
55	ESTIMATED_TIME_TOBE_ACTIVE_FB_ON_END_VELOCITIES_RECALCULATION	

Table 8: Boolean Axis Parameters table

Index	Parameter description	Explanation
1	MMC_AXIS_MODE_PARAM	Profile mode of the Axis: 0 - NC. 1 - Distributed. 2 - Virtual.
2	MMC_AXIS_OP_MOTION_MODE_PARAM	Operation mode of the Axis: (-1) - OPM402_NO (1) - OPM402_PROFILE_POSITION_MODE (2) - OPM402_VELOCITY_MODE (3) - OPM402_PROFILE_VELOCITY_MODE (4) - OPM402_TORQUE_PROFILE_MODE (6) - OPM402_HOMING_MODE (7) - OPM402_INTERPOLATED_POSITION_MODE (8) - OPM402_CYCLIC_SYNC_POSITION_MODE (9) - OPM402_CYCLIC_SYNC_VELOCITY_MODE (10)- OPM402_CYCLIC_SYNC_TORQUE_MODE

3	MMC_AXIS_STATE_PARAM	Axis state parameter (bitmask): NC_AXIS_DONE_MASK 0x80000000 NC_AXIS_VALID_MASKx40000000 NC_AXIS_ERROR_MASK 0x20000000 NC_GROUP_STANDBY_MASK 0x00020000 NC_GROUP_DISABLED_MASK 0x00010000 NC_GROUP_HOMING_MASK 0x00008000 NC_GROUP_ERROR_STOP_MASK 0x00004000 NC_GROUP_MOVING_MASK 0x00002000 NC_GROUP_STOPPING_MASK 0x00001000 NC_AXIS_BUSY_MASK 0x00000800 NC_AXIS_ERROR_STOP_MASK 0x00000400 NC_AXIS_DISABLED_MASK 0x00000200 NC_AXIS_STOPPING_MASK 0x00000100 NC_AXIS_STAND_STILL_MASK 0x00000080 NC_AXIS_DISCRETE_MOTION_MASK 0x00000040 NC_AXIS_CONTINUOUS_MOTION_MASK 0x00000020 NC_AXIS_SYNCHRONIZED_MOTION_MASK 0x00000010 NC_AXIS_HOMING_MASK 0x00000008 NC_AXIS_CONSTANT_VELOCITY_MASK 0x00000004 NC_AXIS_ACCELERATING_MASK 0x00000002 NC_AXIS_DECELERATING_MASK 0x00000001 NC_AXIS_USER_RELEVANT_MASK 0x0FFFFFF8 NC_AXIS_MOTION_STATUS_MASK (NC_AXIS_CONSTANT_VELOCITY_MASK|NC_AXIS_ACCELERATING_MASK|NC_AXIS_DECELERATING_MASK)
4	MMC_AXIS_GROUP_ID_PARAM	Group ID as defined in the resource file
5	MMC_DRIVE_ID_PARAM	Axis ID as defined in the resource file
14	MMC_ACT_VELOCITY_PARAM	Actual velocity of drive (like NC_REC_ACTUAL_POS)
17	MMC_CONSTANT_VELOCITY_PARAM	Indication whether the drive in constant velocity state or not. 1 - In constant velocity. 0-Not in constant velocity.
21	MMC_ACCELERATING_PARAM	Indication whether the drive in accelerating state or not. 1-In accelerating. 0-Not in accelerating.
25	MMC_DECELERATING_PARAM	Indication whether the drive in deceleration state or not. 1-In deceleration 0-Not in deceleration
28	MMC_SPATIAL_OPTION_PARAM	This is a spatial option of ACS profiler line calculation. 0-Square root length. 1-Maximum length.

33	MMC_I_COMM_EV_USR_1_PARAM	Value of UI[] vector within specific index as defined in PDO3\4
34	MMC_I_COMM_EV_USR_1_AUX_PARAM	Value of UI[] vector within specific index of ["USR_1_PARAM" + 1]
37	MMC_I_COMM_EV_USR_2_PARAM	Value of UI[] vector within specific index as defined in PDO3\4
38	MMC_I_COMM_EV_USR_2_AUX_PARAM	Value of UI[] vector within specific index of ["USR_2_PARAM" + 1]
41	MMC_I_COMM_EV_USR_3_PARAM	Value of UI[] vector within specific index as defined in PDO3\4
42	MMC_I_COMM_EV_USR_3_AUX_PARAM	Value of UI[] vector within specific index of ["USR_3_PARAM" + 1]
45	MMC_I_COMM_EV_USR_4_PARAM	Value of UI[] vector within specific index as defined in PDO3\4
46	MMC_I_COMM_EV_USR_4_AUX_PARAM	Value of UI[] vector within specific index of ["USR_4_PARAM" + 1]
50	ERROR_CORRECTION_PARAM	
52	MOTOR_ON_CMD_MAX_TIMEOUT_MS	
56	MMC_MAX_CURRENT_PARAM	

Table 9: Boolean Group Parameters table

Index	Parameter description	Explanation
4	GROUP_ID	
28	SPATIAL_OPTION	

Table 10: Real Axis Parameters table

Index	Parameter description	Explanation
10	MMC_SET_POSITION_PARAM	Desired position of movement (like NC_REC_DESIRED_POS)
11	MMC_ACT_POSITION_PARAM	Actual position of drive (like NC_REC_ACTUAL_POS)
12	MMC_AIM_POSITION_PARAM	Position of last movement function block
13	MMC_SET_VELOCITY_PARAM	Desired velocity (like NC_REC_DESIRED_VEL)
35	MMC_F_COMM_EV_USR_1_PARAM	Value of UF[] vector within specific index as defined in PDO3\4
36	MMC_F_COMM_EV_USR_1_AUX_PARAM	Value of UF[] vector within specific index of ["USR_1_PARAM" + 1]

Index	Parameter description	Explanation
39	MMC_F_COMM_EV_USR_2_PARAM	Value of UF[] vector within specific index as defined in PDO3\4
40	MMC_F_COMM_EV_USR_2_AUX_PARAM	Value of UF[] vector within specific index of ["USR_2_PARAM" + 1]
43	MMC_F_COMM_EV_USR_3_PARAM	Value of UF[] vector within specific index as defined in PDO3\4
44	MMC_F_COMM_EV_USR_3_AUX_PARAM	Value of UF[] vector within specific index of ["USR_3_PARAM" + 1]
47	MMC_F_COMM_EV_USR_4_PARAM	Value of UF[] vector within specific index as defined in PDO3\4
48	MMC_F_COMM_EV_USR_4_AUX_PARAM	Value of UF[] vector within specific index of ["USR_4_PARAM" + 1]
57	MMC_LIMIT_STOP_DECELERATION	
58	MMC_LIMIT_STOP_JERK	
63	MMC_TARGET_RADIUS	
64	MMC_TARGET_TIME	
65	MMC_PROFILE_TIME	
66	MMC_OVERALL_MOTION_TIME	
67	MMC_MAX_TRACKING_ERROR_POSITION	
68	MMC_MAX_TRACKING_ERROR_TIME	
69	MMC_DRIVE_TRACKING_ERROR	
70	MMC_GMAS_TRACKING_ERROR	
71	MMC_END_MOTION_REASON	
72	MMC_AXIS_ERROR_MASK	
73	MMC_LAST_DRIVE_EMERGENCY	
74	MMC_AXIS_ASYNC_ERROR_CODE	
75	MMC_FB_DEPTH	
76	MMC_ANALOG_INPUT	
77	MMC_EMCY_SET_ERR	

Table 11: Real Global Parameters table

Index	Parameter description	Explanation
7	Not in use	
59	MMC_SET_VECTOR_VELOCITY_PARAM	
62	MMC_MCS_S_DIRECTION	

Table 12: Real Group Parameters table

Index	Parameter description	Explanation
15	MMC_MAX_VELOCITY_PARAM	Maximum allowed velocity
16	Not in Use	
18	MMC_SET_ACCELERATION_PARAM	Desired acceleration of movement (like NC_REC_VECT_AC)
19	MMC_MAX_ACCELERATION_PARAM	Maximum allowed acceleration
20	SW_MAX_AC	
22	MMC_SET_DECELERATION_PARAM	Desired deceleration of movement (like NC_REC_VECT_AC)
23	MMC_MAX_DECELERATION_PARAM	Maximum allowed deceleration
24	Not in Use	
26	MMC_MAX_JERK_PARAM	Maximum allowed jerk
27	Not in Use	
31	MMC_SW_LIMIT_HIGH_POS_PARAM	Positive hight limit of the G-MAS
32	MMC_SW_LIMIT_LOW_POS_PARAM	Negative hight limit of the G-MAS
51	S_FACTOR_FOR_POLYNOMIAL_TRANSITION	
60	MMC_MCS_SW_LIMIT_LOW_POS_ARRAY	
61	MMC_MCS_SW_LIMIT_HIGH_POS_ARRAY	

16.2. Future Supported function Blocks

The following function blocks will be supported at a future time.

16.2.1. MMC_MoveContinuous

Commands a controlled motion of a specified relative distance ending with the specified velocity.

```
int MMC_MoveContinuousCmd(
IN MMC_CONNECT_HNDL hConn,
IN MMC_AXIS_REF_HNDL hAxisRef,
IN MMC_MOVECONTINUOUS_IN* pInParam,
OUT MMC_MOVECONTINUOUS_OUT* pOutParam
);
```

Motion Mode	NC - Supported	Distributed - ?
Source	GMAS\includes\MMC_PLCopen_single_API.h	

Function Parameters

hConn

[IN] Connection handle input using hConn, where MMC_CONNECT_HNDL is the Connection Handle Type described in section **10.2.2 MMC_CONNECT_HNDL on page 10-533**. It should be noted that this connection handle is common throughout all G-MAS functions. This connection handle is returned by Init Connection command. If error, returns -1 and a MMC_LIB_API error with further details.

hAxisRef

Axis/group reference handle type returned by GetAxisRef command

pInParam

Points to the **MMC_MOVECONTINUOUS_IN** input data structure using the MMC_MoveContinuous function.

pOutParam

Points to the **MMC_MOVECONTINUOUS_OUT** output structure receiving information as a result of calling the MMC_MoveContinuous function.

Remarks

If the commanded position is reached and no new motion command is placed in the buffer, the axis continues to run with the specified *fEndVelocity*.

Continuous Motion refers to a state where it will not stop by itself.

With the buffered mode, this function block can be replaced by the combination of MMC_MoveRelative and MMC_MoveVelocity.

Scope

MMC_MoveContinuous will only operate from Standstill or Discrete Motion. It will not operate from Stopping, Homing, ErrorStop, or Disabled. Refer to the State diagram in **Figure** 5-1.

MMC_MOVECONTINUOUS_IN Structure

```
typedef struct{
double dbDistance;
float fVelocity;
float fEndVelocity;
float fAcceleration;
float fDeceleration;
float fJerk;
MC_BUFFERED_MODE_ENUM eBufferMode;
unsigned char ucExecute;
}MMC_MOVECONTINUOUS_IN;
```

Parameters

dbDistance

Relative distance for the motion. Any -ve or +ve double values in technical unit [u].

fVelocity

Value of the maximum velocity (not necessarily reached). Any positive float value in u/s.

fEndVelocity

Value of the end velocity. Signed value. Any positive float value in u/s.

fAcceleration

Value of the acceleration (increasing energy of the motor). Any positive float value in u/s^2

fDeceleration

Float value of the deceleration when stopping (decreasing energy of the motor). Any positive float value in u/s^2

fJerk

Float value of the Jerk. Any positive value in u/s^3

eBufferMode

MC_BUFFERED_MODE_ENUM defines the behavior of the axis. Enumerator modes are as follows:

MC_ABORTING_MODE	= 1
MC_BUFFERED_MODE	= 2
MC_BLENDING_LOW_MODE	= 3
MC_BLENDING_PREVIOUS_MODE	= 4
MC_BLENDING_NEXT_MODE	= 5
MC_BLENDING_HIGH_MODE	= 6

Aborting Default mode without buffering. The next function block aborts an ongoing motion and the command affects the axis immediately.

	The buffer is cleared
Buffered	The next function block affects the axis as soon as the previous movement is completed.
BlendingLow	The next function block controls the axis after the previous function block has finished (equivalent to buffered), but the axis will not stop between the movements. The velocity is blended with the lowest velocity of both commands (1 and 2) at the first end-position (1).
BlendingPrevious	Blending with the velocity of function block 1 at the end-position of this block
BlendingNext	Blending with the velocity of function block 2 at end-position of function block1
BlendingHigh	Blending with highest velocity of function block 1 and function block 2 at end-position of function block1.

ucExecute

Start the execution command (Relevant only for future IEC or PLC programming). Boolean TRUE/FALSE values.

MMC_MOVECONTINUOUS_OUT Structure

```
typedef struct{
unsigned int uiHndl;
unsigned short usStatus;
short usErrorID;
}MMC_MOVECONTINUOUS_OUT;
```

Parameters

uiHndl

Returned function block handle. Integer with any +ve value

usStatus

Bitwise returned command status with the following values:

Aborted
Done
CommandError

usErrorID

Returned command error ID. Signals where an error has occurred within function block. Refer to the errors listed in sections **6.2 G-MAS Error IDs**, and **6.7 NC Profiler Error IDs on page 6-417** - **6-453**. Displays an error code as -ve or +ve integers.

Figure 16-1 describes the function block for MMC_MoveContinuous

Figure 16-1: MMC_MoveContinuous function block

16.2.1.2. Function Block Code Example

```
int rc ;
MMC_MOVECONTINUOUS_IN       stMoveCont_in;
MMC_MOVECONTINUOUS_OUT      stMoveCont_out;
//
// Inserting the structure parameters:
stMoveCont_in.fAcceleration= 100000.0; // Value of the acceleration
stMoveCont_in.fDeceleration  = 100000.0; // Value of the deceleration
stMoveCont_in.fJerk = 2000.0;// Value of the Jerk
stMoveCont_in.eBufferMode = MC_BUFFERED_MODE;  // MC_BUFFERED_MODE_ENUM Defines the behavior
of the axis
stMoveCont_in.dbDistance  = 100000.0; // Target position for the motion
stMoveCont_in.fVelocity= 5000.0;// Velocity in
stMoveCont_in.fEndVelocity= 3000.0;// Value of the end velocity
stMoveCont_in.ucExecute= 1;
//
rc = MMC_MoveContinuousCmd (hConn, iAxisRef, &stMoveCont_in, &stMoveCont_out);
if (rc != 0)
{
 HandleError() ;
}
```

16.2.1.3. Implementation Example

These two sampling traces describe why an EndVelocityDirection input is necessary and how it effects the movement.

1. fEndVelocity positive:

Figure 16-2: MMC_MovedContinuous with a positive fEndVelocity - Example

2. fEndVelocity negative:

Figure 16-3: MMC_MovedContinuous with a negative fEndVelocity - Example

The following is an example using MC_MoveContinuous:

Figure 16-4: Example of MMC_MoveContinuous with Timing diagram

Chapter 17: Index

ACS 1-18
Active range 14-1045
AddAxisToGroup 15-1152
Administrative function blocks 5-363
Advanced trigger options 14-1044
Angle Mode 5-266
API library 1-17
API services and operations 10-533
AppendPointsToPVTTable 15-1228
Axis 1-18
Axis parameters 5-64, 16-1258
Axis Status 5-69
Axis Status Bit Masks 5-69
AxisLink 15-1130
AxisUnLink 15-1131
Blending 1-18
Buffered Errors 6-419
Bulk Reading functions 8-491, 9-510
BulkRead 15-1222
CallbackFunc 15-1192
CAN 1-18
CAN master (or client) 11-742
CANbus drive communication function blocks 11-748
CANbus motion (DS-402) communication 11-741
CancelPDO 15-1113
CancelVirtualEncoder 15-1126
CANopen 11-741
CANopen DS-402 standard 2-33, 11-742
CANopen protocol 11-741
ChangeDefaultPDOConfig 15-1114
CiA 1-18
Circular Modes 5-264
ClearConnectionReg 15-1179
CMMCBulkRead 15-1221
COB 1-18
COB-ID 1-18
COB-IDs 11-741
CoE 1-18
Coil tables 11-715
Config 15-1222
ConfigBulkRead 15-1176
ConfigPDO 15-1110
ConfigPDOEventMode 15-1113
ConfigVirtualEncoder 15-1125
ConnectIPC 15-1184
ConnectIPCEx 15-1185
Connectivity and configuration function blocks 11-669
ConnectRPC 15-1186
ConnectRPCEx 15-1187
Contour curve 1-18
Coordinate system 1-18
Coordinated Motion 5-240
Corner deviation 1-18
Corner distance 1-18
Create 15-1247
CreateSYNCTimer 15-1174
Data recording 14-1044
Data Recording 14-1045
DestroySYNCTimer 15-1175
Development / Host PC 1-17
dhcp – Static / Dynamic IP 4-45
Direction 1-18
DisableMotionEndedEvent 15-1076
Distributed 2-31
Download firmware function blocks 10-535
Drive 1-18
Drive (PDS - Power Drive System) 11-742
DS-401 drive communication function blocks 11-963
DS-401 drive communications 11-963
DS-402 motion modes 2-33
DS-402 operation modes 2-31
EAS 2-29
EipCheckDevTagReply 11-948
EipCheckReply 15-1237
EIPCloseSession 11-952, 15-1232
EipCreate 11-954, 15-1232
EipDestroy 11-956, 15-1233
EipGetAdpTagRefByName 11-923
EipGetAssembly 11-936
EipGetAssemblyRefByInstance 11-930
EipGetAssemblyRefByName 11-932
EipGetData 15-1238
EipGetDevTag 11-942
EipGetDevTagRefByName 11-938
EipGetTag 15-1236
EipOpenSession 11-950, 15-1233
EipReadAdpTag 11-928
EipReadDevTagData 11-944
EipSetAssembly 11-934
EipSetDevTag 11-940
EipSetTag 15-1235
EipSyncGetDevTag 11-946
EipTagInit 15-1235
EipWriteAdpTag 11-925
Elmo Gold Maestro controller 1-16
ElmoCallAsync 15-1122
ElmoExecute 15-1123
ElmoGetArray 15-1255
ElmoGetAsyncArray 15-1254
ElmoGetAsyncFloatArray 15-1118
ElmoGetAsyncFloatParam 15-1116
ElmoGetAsyncIntArray 15-1117
ElmoGetAsyncIntParam 15-1116
ElmoGetAsyncParameter 15-1254
ElmoGetParameter 15-1256
ElmoGetReply 15-1124
ElmoGetSyncArray 15-1121
ElmoGetSyncParam 15-1120
ElmoIsReplyAwaiting 15-1123
ElmoReadData 15-1257
ElmoSetArray 15-1252
ElmoSetAsyncArray 15-1119, 15-1250
ElmoSetAsyncParam 15-1115
ElmoSetAsyncParameter 15-1251
ElmoSetParameter 15-1253
EnableMotionEndedEvent 15-1076
EoE 1-18
Error Correction functions 7-475
Error Handling 6-417
EtherCAT 2-33
EtherCAT drive communication function blocks 11-833
EtherCAT Gateway 11-832
EtherCAT protocol 11-828
EtherCAT with G-MAS 11-831
Ethernet for Control Automation Technology (EtherCAT) 11-828
EtherNetIP communication functions 11-922
Events enumerator 12-1017
Events function blocks 12-1025
Excluding Triggers 14-1045
FIFO 1-18

FIFO index 11-890
Firmware download errors 10-535
FoE 1-19
Function blocks 2-27
Function Parameters 1-23
Get Function – Asynchronous Mode 11-890
GetActualPosition 15-1102
GetActualTorque 15-1103
GetActualVelocity 15-1102
GetAxisByName 15-1078
GetAxisError 15-1105
GetBoolParameter 15-1081, 15-1131
GetCommDiagnostics 15-1196
GetConnectionReg 15-1175, 15-1179
GetDigInput[s] 15-1105
GetDigOutputs 15-1106
GetDigOutputs32Bit 15-1106
GetFbDepth 15-1077
GetGlobalBoolParameter 15-1189
GetGlobalParameter 15-1190
GetGroupAxisByName 15-1078
GetNetworkInfo 15-1198
GetOpMode 15-1100
GetParameter 15-1082, 15-1132
GetPDOInfo 15-1219
GetSyncTime 15-1172
G-MAS 1-16
G-MAS Axis (Node) State Management mechanism 11-743
G-MAS Error IDs 6-420
G-MAS general communication and API architecture 2-26
G-MAS modes 2-31
G-MAS over standard 1-16
G-MAS Personality Functions 13-1041
GoS 1-19
Group 1-19
Group motion commands 5-61
Group of axes 2-31
Group of Axes 5-61
GroupDisable 15-1149
GroupEnable 15-1148
Group-FB 1-19
GroupHalt 15-1155
GroupReadActualPosition 15-1153
GroupReadActualVelocity 15-1150
GroupReadError 15-1151
GroupReadStatus 15-1148
GroupReset 15-1149
GroupSetOverride 15-1146
GroupSetPosition 15-1147
GroupStop 15-1154
Halt 15-1103
Home 15-1089
HomeDS402 15-1090
Homing Functions 5-87
Homing Procedures 5-88
Host communications 11-715
How to use the detailed information within each function block 1-22
HPT 1-19
IEC 61158 11-828
Including Triggers 14-1046
InitPVTTable 15-1226
Interpreter Command Functions 11-890
IO 1-19
ipaddr – GMAS IP Address 4-46
IPC 1-19
ipmask – GMAS IP Subnet Mask 4-46
LoadPVTTable 15-1227
Masking 1-19
Max Axes and Node numbers 2-32
MbusIsRunning 15-1203
MbusReadCoilsTable 15-1204
MbusReadHoldingRegisterTable 15-1202
MbusReadInputsTable 15-1206
MbusStartServer 15-1200
MbusStopServer 15-1201
MbusWriteCoilsTable 15-1205
MbusWriteHoldingRegisterTable 15-1203
MC_FinishHoming 5-96
MCS 1-19
MMC_AddAxisToGroup 5-364
MMC_AppendPointsToTable 9-525
MMC_AxisLink 5-234
MMC_AxisUnLink 5-237
MMC_BeginRecording 14-1054
MMC_CancelGeneralRPDO3 11-964
MMC_CancelGeneralRPDO4 11-967
MMC_CancelGeneralTPDO3 11-970
MMC_CancelGeneralTPDO4 11-973
MMC_CancelParamEvPDO3 11-751
MMC_CancelParamEvPDO4 11-754
MMC_CancelVirtualEncoder 11-749
MMC_CfgRegParamEvPDO3 11-757
MMC_CfgRegParamEvPDO4 11-761
MMC_CfgUserParamEvPDO3 11-765
MMC_CfgUserParamEvPDO4 11-769
MMC_ChangeDefaultPDOConfiguration 11-774
MMC_ChangeToOperationMode 10-541
MMC_ChangeToPreOPMode 10-538
MMC_ChngOpMode 11-777
MMC_ClearEventsMask 12-1026
MMC_ClearNodeFbList 10-544
MMC_CloseConnection 10-550
MMC_CloseUdpChannel 11-670
MMC_CmdStatus 10-547
MMC_Config 10-552
MMC_ConfigBulkRead 8-492
MMC_ConfigEventModePDO3 11-780
MMC_ConfigEventModePDO4 11-783
MMC_ConfigGeneralRPDO3 11-976
MMC_ConfigGeneralRPDO4 11-979
MMC_ConfigGeneralTPDO3 11-982
MMC_ConfigGeneralTPDO4 11-985
MMC_ConfigVirtualEncoder 11-786
MMC_CONNECT_HNDL 10-536
MMC_Connection_Param_Struct 10-533
MMC_CreatePersonalityParameters 13-1042
MMC_CreateSYNCTimer 10-555
MMC_DestroySYNCTimer 10-557
MMC_DisableDS401DIChangedEvent 11-988
MMC_DisableErrorCorrTable 7-485
MMC_DisableEthercatConfigMode 11-834
MMC_DisableMotionEndedEvent 12-1029
MMC_DOWNLOAD_TYPE_ENUM 10-536
MMC_DownloadFoE 10-558
MMC_DownloadVersion 10-668
MMC_Dwell 10-563
MMC_ECATIODisableDIChangedEvent 11-839
MMC_ECATIOEnableDIChangedEvent 11-842
MMC_ECATIOReadAnalogInput 11-848
MMC_ECATIOReadDigitalInput 11-845
MMC_ECATIOWriteAnalogOutput 11-851
MMC_ECATIOWriteDigitalOutput 11-854
MMC_ElmoCall 11-911
MMC_ElmoExecuteLabel 11-892
MMC_ElmoGetArray 11-899
MMC_ElmoGetArrayAndRetrieveData 11-901
MMC_ElmoGetParameter 11-897
MMC_ElmoGetParameterAndRetrieveData 11-903
MMC_ElmoQueryOperationFIFOIndex 11-907
MMC_ElmoQueryOperationFIFOIndexReset 11-910
MMC_ElmoQueryOperationFIFORetrieveData 11-908

MMC_ElmoSetArray 11-905
MMC_ElmoSetParameter 11-895
MMC_EnableDS401DIChangedEvent 11-991
MMC_EnableErrorCorrTable 7-479
MMC_EnableEthercatConfigMode 11-837
MMC_EnableMotionEndedEvent 12-1032
MMC_Exit 10-566
MMC_FreeFbStat 10-569
MMC_GetActiveAxesNum 10-652
MMC_GetActiveVectorsNum 10-572
MMC_GetAxisByCanId 11-789
MMC_GetAxisByName 10-587
MMC_GetBulkUploadData 11-825
MMC_GetBulkUploadStatus 11-822
MMC_GetCommDiagnostics 11-870
MMC_GetCommStatistics 11-857
MMC_GetCyclesCounter 10-657
MMC_GetDefGateway 11-673
MMC_GetDhcp 11-676
MMC_GetEnquireFbStatus 10-584
MMC_GetErrorCodeDescriptionByID 10-575
MMC_GetErrorTableStatus 7-482
MMC_GetEthercatCommStatistics 11-861
MMC_GetEventsMask 12-1035
MMC_GetFBDepth 5-149
MMC_GetFoEStatus 10-578
MMC_GetGMASOperationMode 10-598
MMC_GetGroupByName 10-590, 10-593
MMC_GetIpAddr 11-679
MMC_GetIpMask 11-682
MMC_GetPDOInfo 11-792
MMC_GetReactorStatistics 11-874
MMC_GETREACTORSTATISTICS 5-148
MMC_GetResList 10-604
MMC_GetResSnapshot 10-607
MMC_GetServerIp 11-685
MMC_GetStatusRegister 10-601
MMC_GetSyncTime 11-795
MMC_GetTableIndex 9-529
MMC_GetTotalFbDepth 5-152
MMC_GetVerPath 10-668
MMC_GetVersion 10-610
MMC_GetVersionEx 10-613
MMC_GlobalReadBoolParameter 5-177
MMC_GlobalReadParameter 5-194
MMC_GlobalWriteBoolParameter 5-219
MMC_GlobalWriteParameter 5-231
MMC_GroupDisable 5-367
MMC_GroupEnable 5-370
MMC_GroupHalt 5-281
MMC_GroupReadActualPosition 5-373
MMC_GroupReadActualVelocity 5-376
MMC_GroupReadError 5-379
MMC_GroupReadStatus 5-382
MMC_GroupReset 5-385
MMC_GroupSetOverride 5-388
MMC_GroupSetPosition 5-393
MMC_GroupStop 5-277
MMC_GroupWriteParameter 5-405, 5-408, 5-411, 5-414
MMC_Halt 5-78
MMC_Home 5-82
MMC_HomeDS402 5-97
MMC_InitConnection 10-616
MMC_InitTable 9-510
MMC_IpcInitConnection 10-618
MMC_IsEthercatConfigMode 11-877
MMC_LoadErrorCorrTable 7-476
MMC_LoadParam 10-620
MMC_LoadTableFromFile 9-515
MMC_MbusIsRunning 11-717
MMC_MbusReadCoilsTable 11-720
MMC_MbusReadHoldingRegisterTable 11-723
MMC_MbusReadInputsTable 11-726
MMC_MbusStartServer 11-729
MMC_MbusStopServer 11-732
MMC_MbusWriteCoilsTable 11-735
MMC_MbusWriteHoldingRegisterTable 11-738
MMC_MCS_Info_Struct 5-239
MMC_MCS_Kin_Ref_Struct 5-239
MMC_MoveAbsolute 5-102
MMC_MoveAbsoluteRepetitive 5-126
MMC_MoveAdditive 5-108
MMC_MoveAdditiveRepetitive 5-136
MMC_MoveCircularAbsolute 5-285
MMC_MoveCircularAbsoluteAngle 5-312
MMC_MoveCircularAbsoluteBorder 5-299
MMC_MoveCircularAbsoluteCenter 5-293
MMC_MoveCircularAbsoluteRadius 5-305
MMC_MoveContinuous 16-1263
MMC_MoveLinearAbsolute 5-317
MMC_MoveLinearAbsoluteRepetitive 5-341
MMC_MoveLinearAdditive 5-331
MMC_MoveLinearAdditiveEx 5-336
MMC_MoveLinearRelative 5-324
MMC_MoveLinearRelativeRepetitive 5-346
MMC_MovePath 5-355
MMC_MoveRelative 5-114
MMC_MoveRelativeRepetitive 5-131
MMC_MoveTable 9-521
MMC_MoveVelocity 5-120
MMC_NetworkInfo 11-688
MMC_NetworkScan 11-694
MMC_NodesInfo 11-669
MMC_OpenUdpChannel 11-697
MMC_PathSelect 5-351, 15-1159
MMC_PathUnselect 5-360
MMC_PDOGeneralRead 11-798
MMC_PDOGeneralWrite 11-801
MMC_PerformBulkRead 8-500
MMC_PositionProfile 5-159
MMC_Power 5-155
MMC_Read DS401DIGroup 11-994
MMC_ReadActualPosition 5-162
MMC_ReadActualTorque 5-165
MMC_ReadActualVelocity 5-168
MMC_ReadAxisError 5-171
MMC_ReadBoolParameter 5-174
MMC_ReadDigitalInput(s) 5-180
MMC_ReadDigitalOutputs 5-185
MMC_ReadDigitalOutputs32Bit 5-188
MMC_ReadDownloadVersionStatus 10-668
MMC_ReadDS401DInput 11-997
MMC_ReadGroupOfParameters 10-665
MMC_ReadParameter 5-191
MMC_ReadStatus 5-60, 5-197
MMC_ReceiveCANRawData 11-804
MMC_RecStatus 14-1064
MMC_RemoveAxisFromGroup 5-397
MMC_Reset 5-200
MMC_ResetCommDiagnostics 11-880
MMC_ResetCommStatistics 11-883
MMC_ResetMultiAxisControl 10-627
MMC_ResExportFile 10-630
MMC_ResImportFile 10-633
MMC_RpcInitConnection 10-623
MMC_RpcInitConnectionEx 10-625
MMC_SaveParam 10-636
MMC_SendCANRawData 11-807
MMC_SendCmd 11-810
MMC_SendSDO 11-886
MMC_SetDefaultParameters 10-641
MMC_SetDefaultParametersGlobal 10-644

MMC_SetDefGateway 11-700
MMC_SetDhcp 11-703
MMC_SetEnquireFbStatus 10-639
MMC_SetEventsMask 12-1038
MMC_SetHeartBeatConsumer 11-813
MMC_SetIpAddr 11-706
MMC_SetIpMask 11-709
MMC_SetIsToLoadGlobalParams 10-647
MMC_SetKinTransform 5-400
MMC_SetOverride 5-203
MMC_SetPosition 5-207
MMC_SetServerIp 11-712
MMC_SetSyncTime 11-816
MMC_SetVerPath 10-668
MMC_ShowNodeStat 10-649
MMC_StartBulkUpload 11-819
MMC_Stop 5-141
MMC_StopRecording 14-1058
MMC_ToggleConsoleOutput 10-655
MMC_TouchProbeDisable 5-213
MMC_TouchProbeEnable 5-210
MMC_UnloadErrorCorrTable 7-488
MMC_UnloadTable 9-518
MMC_UploadData 14-1061
MMC_UploadDataHeader 14-1067
MMC_WriteBoolParameter 5-216
MMC_WriteDigitalOutputs 5-222
MMC_WriteDigitalOutputs32Bit 5-225
MMC_WriteDS401DOGroup 11-1000
MMC_WriteDS401DOutput 11-1003
MMC_WriteGroupOfParameters 10-660
MMC_WriteParameter 5-228
Modbus communication function blocks 11-716
Modbus interface 11-715
Motion Mode 1-23
Motor 1-19
MoveAbsolute 15-1091
MoveAbsoluteRepetitive 15-1095
MoveAdditive 15-1092
MoveAdditiveRepetitive 15-1097
MoveCircularAbsolute 15-1138
MoveCircularAbsoluteAngle 15-1143
MoveCircularAbsoluteBorder 15-1141
MoveCircularAbsoluteCenter 15-1140
MoveCircularAbsoluteRadius 15-1142
MoveLinearAbsolute 15-1144
MoveLinearAbsoluteRepetitive 15-1156
MoveLinearAdditive 15-1158
MoveLinearRelative 15-1145
MoveLinearRelativeRepetitive 15-1157
MovePath 15-1159
MovePVT 15-1229
MoveRelative 15-1093
MoveRelativeRepetitive 15-1096
MoveVelocity 15-1094
Multiple axes function blocks 5-239
Mutex 1-19
NC Axes 2-31
NC Motions 2-31
NC Profiler caution IDs 6-464
NC Profiler Error IDs 6-456
NC_COMM_EVENT_GROUP 11-743
NC_MCS_Info_Struct 5-275
NC_MCS_Kin_Ref_Struct 5-276
Network communications 11-669
Orientation 1-20
Path 1-20
PathData 1-20
PathDeselect 15-1163
PathSelect 15-1164
PDO 1-20
PDO mapping 11-743
PDOGeneralRead 15-1217
PDOGeneralWrite 15-1218
PDS 1-20
PerformBulkRead 15-1178
PLCMotion API definition 2-33
PLCopen Motion API 1-17
PLCopen specification 5-59
Position 1-20
PositionProfile 15-1098
PowerOff 15-1101
PowerOn 15-1101
Process Data Object (PDO) 11-741
ReadStatus 15-1212
ReceiveFrom 15-1249
Recording Data Signal Bitmask 14-1047
Recording signal variables 14-1047
RegisterConnection 15-1179
RegisterEventCallback 15-1193
RegisterRTE 15-1167
RegisterSyncTimerFunction 15-1194
RegisterWarningClbk 15-1167
Registry tables 11-715
Regular Event Groups 11-744
RemoveAxisFromGroup 15-1137
Reset 15-1211
ResetCommDiagnostics 15-1196
ResetCommStatistics 15-1197
RetreiveSDOUploadAsync 15-1217
RPC 1-20
RPDO 1-20, 11-741
Scara 1-20
Scope 1-23
SDO 1-20
SDO protocol 11-741
SendSDO 15-1213
SendSDODownload 15-1214
SendSDOUpload 15-1215
SendSDOUploadAsync 15-1216
SendTo 15-1248
Service Data Object (SDO) 11-741
servo drives 2-33
SetBoolParameter 15-1079, 15-1129
SetConnectionType 15-1170
SetDefaultHomeDS402Params 15-1087
SetDefaultHomeParams 15-1088
SetDefaultManufacturerParameters 15-1077
SetDefaultParams 15-1086, 15-1135
SetDigOutputs 15-1108
SetDigOutputs32Bit 15-1107
SetGlobalBoolParameter 15-1188
SetHeartBeatConsumer 15-1192
SetIsToLoadGlobalParams 15-1191
SetKinTransform 15-1136
SetMessageFileName 15-1171
SetModbus[LongSwapped][Short] 15-1207
SetOpMode 15-1100
SetOverride 15-1109
SetParameter 15-1080, 15-1128
SetPosition 15-1127
SetPrintErrorFlag 15-1169
SetPrintWarningFlag 15-1169
SetSyncTime 15-1173
SetThrowFlag 15-1168
SetThrowWarningFlag 15-1168
SetTimeout 15-1250
Single Axis 5-59
Single axis administrative function blocks 5-146
Single Axis as Distributed axis 2-31
Single Axis as NC axis 2-31
Single axis function blocks 5-77

Slave servo drive 11-742
Source 1-23
Speed 1-20
standard IEEE 802.3 Ethernet frame 11-828
Stop 15-1104
Sub-parameters 1-24
SYNC 11-746
Synchronization 1-20, 11-746
TCP 1-20
terminology 1-18
ThrowMessage 15-1170
To connect the host system to the G-MAS via USB connection 4-44
TouchProbeDisable 15-1099
TouchProbeEnable 15-1099
TPDO 1-20, 11-741
Tracking 1-20
Trajectory 1-21
Transition and Circular Modes 5-252
Trigger event types 14-1044
Trigger modes 14-1052
TX PDOs 11-743
Type Definitions 10-536
UnloadPVTTable 15-1229
USB connection 4-44
User Event Groups 11-744
User Float 11-746
User Integer 11-746
Velocity 1-21